

2<sup>ème</sup> EDITION

**Led**  
hors série

# ETUDES AUTOUR DU 6809

(CONSTRUCTIONS ET LOGICIELS)



Diffusion



EYROLLES

Claude Vicidomini



# **ETUDES AUTOUR DU 6809**

**(CONSTRUCTIONS ET LOGICIELS)**

**2<sup>e</sup> édition**



Diffusé par EYROLLES  
61, bd Saint-Germain, 75240 Paris Cédex 05

**Société éditrice : Editions Fréquences** Siège Social : 1, bd Ney, 75018 Paris - Tél. : (1) 40.36.01.97 + - SA au capital de 1.000.000 F - Président-Directeur Général : Edouard Pastor.  
**LED Initiation - Etudes autour du 6809 - 140 F** - Commission paritaire : 64949 - Directeur de la publication : Edouard Pastor - Tous droits de reproduction et d'adaptation réservés textes et photos pour tous pays - LED est une marque déposée ISSN 0753-7409.  
Rédaction : Claude Vicidomini.  
Photocomposition, mise en page, photogravure et réalisation : Edi'Systèmes.  
Impression : Berger-Levrault - Nancy.



## Chapitre 1

# De la logique câblée au microprocesseur

### Introduction au microprocesseur

#### L'être physique : l'homme

Le génie créatif de l'homme est probablement sans limite ; voici qu'il a inventé, il y a maintenant plus de dix années, le microprocesseur qui nous ouvre toutes grandes les portes de l'ère post-industrielle.

Le microprocesseur est ce que l'on pourrait appeler le «cerveau profond» d'une machine intelligente.

#### Notion d'environnement

Nous sommes en permanence baignés dans un environnement qui nous communique, par l'intermédiaire d'organes, le contexte dans lequel nous nous trouvons.

Que nous soyons endormis ou éveillés, notre cerveau effectue un scanning permanent. Il est toujours en éveil mais il nous cachera certains résultats, jugeant qu'ils ne sont pas assez importants pour être mis au niveau de notre conscience.

Par exemple : la station debout exige des efforts permanents des muscles de notre corps pour garder l'équilibre. Notre oreille interne fournit au cerveau le sens de la pesanteur. Si nos jambes ressentent le poids de notre corps, nous sommes donc debout ; si notre postérieur ressent ce poids, nous sommes assis.

En fait, notre cerveau effectue des opérations logiques. Dans l'exemple ci-dessus, il s'agit d'un ET, les notions de verticalité et de poids s'appellent des paramètres, ainsi Station debout (d) = Vertical (v) ET Poids (p) ou  $d = v.p.$

Il est bien évident que l'association de plusieurs paramètres (plus de 2) nous place dans un certain contexte.

Il peut y avoir contradiction et, dans ce cas, nous ne savons plus où nous en sommes, ce qui entraîne, chez l'homme, des réactions tout à fait semblables à un malaise. Par exemple, le fait de se trouver dans l'espace, sans poids et sans référence de pesanteur amène notre cerveau à prendre de nouvelles habitudes... mais que la pesanteur revienne brutalement et c'est le malaise qui entraîne des vomissements : le cerveau ne sait plus quels sont les bons paramètres à prendre en compte ! Autrement dit :  $d \neq v.p.$  L'exemple que nous venons de citer constitue une expérience très importante de Patrick Baudry dans le vol orbital de la navette spatiale.

D'autres contextes entraînent d'autres formulations logiques : on peut introduire le OU logique (la saveur de ce met me laisse à penser qu'on a un peu forcé sur l'épice OU que le plat est naturellement épicé OU les deux à la fois), le OU

exclusif (uniquement l'un OU uniquement l'autre). Toutes ces formulations de type booléenne peuvent être fort complexes et notre cerveau est parfaitement capable de trouver la ou les racines de toute équation.

#### Notion d'organes

Pour que notre cerveau soit capable de sûpputer et de prendre des décisions pour un contexte donné, il faut le brancher à des organes capables de recueillir ou d'envoyer toutes sortes d'informations..., et elles sont variées :

- l'intensité lumineuse est une première information : elle se cantonne dans le cas de l'homme, dans le spectre visible, l'œil est l'organe essentiel de la vue. Notre peau, dans une moindre mesure, capte des rayonnements infra-rouges nous donnant ainsi la sensation du chaud et du froid.

- le bruit, interprété ici dans sa forme générale (des vibrations de fréquence très basse: infrasons ou ultrasons), est un support extraordinaire qui nous permet de comprendre (l'ouïe) et de nous faire comprendre (la voix). Ce bruit peut être harmonieux (le chant) ou franchement désagréable (le son d'une guitare désaccordée ou les réacteurs d'un avion en phase de décollage). Notons que dans ce dernier cas, notre corps se met en vibration en résonance avec les infrasons engendrés par les réacteurs).

- le volume d'un objet nous est communiqué à la fois par nos mains et notre vue : nous savons reconnaître une cuillère ou une fourchette les yeux fermés, nous savons évaluer le poids de ce tome ainsi que ses dimensions toujours les yeux fermés, mais nous n'aurons jamais l'idée de faire de même avec un immeuble de 50 étages !

C'est que là, notre cerveau associé à notre vue, est capable de faire des prodiges d'ingéniosité : puisqu'il est impossible de soupeser cet immeuble, vu ses dimensions, il peut aller chercher en mémoire une information relative à un immeuble d'une taille semblable lue dans un article d'une revue puis négocier le poids réel en fonction des informations recueillies par la vue.

Une autre solution consiste à calculer la masse de l'immeuble en évaluant le poids d'une quantité de maçonnerie connue et en le multipliant par le nombre d'éléments constituant l'immeuble.

D'autres solutions existent encore et l'imagination de notre cerveau est sans limite pour faire face à de telles situations.

#### Notion de mémoire

Nous l'avons vu dans l'exemple ci-dessus, une mémoire est indispensable pour nous permettre de nous souvenir d'événements ayant un rapport avec la situation que nous vivons.

Notre cerveau dispose de plusieurs milliards de ces boîtes aux lettres qui s'enrichissent sans cesse de faits nouveaux. En fait, cette mémoire «boîte aux lettres» nous sert à évaluer, comparer, négocier une situation par rapport à des paramètres préétablis ; considérons l'expérience suivante :

- dans une boîte, disposons 8 boules de couleur noire et 2 boules de couleur blanche, puis fermons le couvercle et, en agitant bien l'ensemble, faisons tomber 5 boules dans une seconde boîte. Le contenu de la deuxième boîte nous donnera aussitôt le contenu de la première par simple déduction logique.

Si nous y trouvons 5 boules noires, il reste donc dans la première boîte 3 boules noires et 2 blanches.

Si nous trouvons 2 boules blanches dans la deuxième boîte, il reste donc 5 boules noires dans la première boîte...

Les données de départ étaient :

Paramètre 1 = 2 boules blanches

Paramètre 2 = 8 boules noires

Total = 10 boules = paramètre 1 + paramètre 2

Les données d'arrivée sont donc :

$0 \leq \text{Paramètre 1} \leq 2$

$0 \leq \text{Paramètre 2} \leq 5$

Notre cerveau effectue donc une simple opération arithmétique amenant le résultat final mais il fallait bien qu'il demande à notre mémoire «boîte aux lettres» les paramètres de départ pour en tirer la conclusion qui s'impose.

### Notion d'ordonnancement ou de séquençement

Nous en arrivons au point final et crucial du fonctionnement d'une machine humaine : toutes les données fournies par notre environnement ainsi que toutes celles que nous lui fournissons doivent être «temporisées». Une unité de séquençement se chargera de l'affaire.

Sa présence est indispensable car sans elle tout serait mélangé, sans queue ni tête ; bref, ce serait la confusion totale !

Imaginez-vous donc en train de vous servir un verre d'eau... sans verre, ou prendre votre soupe avec une fourchette... les exemples abondent, il suffit d'ailleurs de citer le cas des fous qui ne peuvent plus négocier leurs actes d'une façon logique.

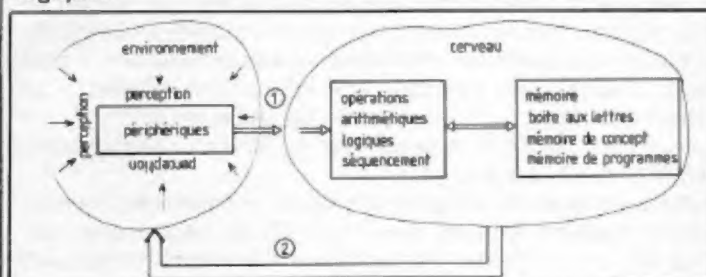


Fig. 1 : Trajet 1 : perception de phénomènes extérieurs.  
Trajet 2 : action s'il y a lieu.

### Résumé

Une machine intelligente, en l'occurrence un être humain, dispose d'un cerveau comprenant :

- une unité arithmétique et logique effectuant toutes les opérations logiques (OU, ET, NON...) et arithmétiques (+, -, ×, ÷...)
- une mémoire «mémorisant» des données (exemple des

boules, exemple de l'immeuble) ou exécutant des tâches selon un algorithme préétabli (exemple : la station debout exige l'action de certains muscles du corps pour ne pas perdre l'équilibre...)

- des organes périphériques (mains, yeux, nez...) permettant de communiquer avec le monde extérieur.

- une unité de séquençement (ou d'ordonnancement) synchronisant des tâches élémentaires par rapport à une unité de temps afin de ne pas provoquer de mélanges ou «patchworks» intempestifs.

En réalité, la puissance de notre cerveau permet d'exécuter plusieurs tâches différentes simultanément (le fait de marcher dans la rue ne nous empêche pas de dialoguer avec notre voisin).

### L'être physique : le robot

On conçoit aisément que le but à atteindre par tout industriel est de réaliser la machine idéale et parfaite.

Le robot idéal sera ce que les auteurs de science fiction appellent un «androïde» autrement dit un être humain fabriqué industriellement par des humains !

Nous n'en sommes pas encore là car l'évolution de l'informatique en est encore à son balbutiement.

En effet, la difficulté de réalisation d'un robot ne réside pas dans sa partie matérielle : on sait fabriquer des mains ayant autant de degrés de liberté que celles de l'être humain, les yeux remplacés ici par des caméras CCD sont plus sensibles que les yeux humains mais leur définition est bien moindre. Il va sans dire que la technologie évolue à grande vitesse dans ce domaine-là et nous verrons bientôt l'arrivée sur le marché de caméras très performantes ; la synthèse et la reconnaissance vocale font actuellement des progrès extraordinaires.

En fait, la principale difficulté réside dans la réalisation du cerveau de ce robot. Il doit être intelligent, autrement dit, il doit être aussi complexe que le cerveau humain, or, il n'existe actuellement aucune technologie opérationnelle capable de concurrencer notre cerveau. Un deuxième problème, et non le moindre, réside dans l'intelligence de ce cerveau : aucun programme, aucun algorithme ne peut simuler le fonctionnement de notre cerveau.

En fait, ces programmes appelés «systèmes experts» en sont, à leurs tous premiers balbutiements, les industries nipponne et américaine se sont lancées dans la réalisation de programmes experts ouvrant grande la porte de l'intelligence artificielle (notons que l'Europe, un peu à la traîne, songe à combler rapidement ce retard).

Les ordinateurs des années 90 auront peu évolué technologiquement mais les logiciels seront très performants car ils seront intelligents, c'est-à-dire capables d'exécuter les ordres du programmeur avec très peu de manipulations.

Nous avons eu l'occasion d'utiliser un programme d'éducation assisté par ordinateur qui utilise un système expert en guise de logiciel et nous avons été surpris de la rapidité et de la simplicité de mise en œuvre du cours que nous voulions mettre au point.

Le plus grand progrès de cette décennie passera donc par cette voie : imaginez en effet qu'il vous sera enfin possible de «jouer» avec votre ordinateur sans avoir besoin de lire



préalablement le mode d'emploi rédigé en trois tomes !

De plus, tous les programmes qui seront vendus sur le marché pourront être exécutés sur votre ordinateur de même que tous vos programmes pourront être exécutés sur un ordinateur de marque concurrente.

Aujourd'hui, nous ne savons pas fabriquer des robots parfaits, mais nous savons par contre fabriquer des éléments disparates dont l'ensemble constituera une machine intelligente appelée «micro-ordinateur».

La principale différence entre ordinateur et micro-ordinateur réside en 2 points :

- La taille : un ordinateur occupe une surface au sol de plusieurs mètres carrés : il utilise une mémoire très importante et peut se raccorder sur plusieurs ordinateurs formant ainsi un réseau pouvant être très complexe. Plusieurs personnes peuvent faire exécuter des tâches différentes sur cet ordinateur sans qu'il soit trop encombré.

En revanche, un micro-ordinateur occupe peu de place (une petite table lui suffit), sa mémoire est réduite bien qu'elle soit capable d'être agrandie de façon notable, il lui sera difficile de former un réseau mais il peut devenir la liaison terminale d'un ordinateur ; plusieurs personnes peuvent exécuter des tâches différentes mais ce nombre de personnes doit être réduit (de 4 à 16) car sa vitesse de travail diminue à chaque ajout d'un utilisateur... dans ce cas précis, le temps d'accès à une information peut quelquefois être très long.

- Le prix : un ordinateur est par définition un système professionnel, car son coût est souvent très supérieur à 600 KF.

Pour ce prix, on a un système très fiable avec une assistance rapide en cas de panne.

Un «bon» micro-ordinateur dans la gamme des professionnels se situe dans une fourchette de prix entre 100 KF et 200 KF, ses performances sont voisines de celles d'un ordinateur bas de gamme et de plus il bénéficie d'une assistance technique très efficace. Il a aussi l'avantage d'être modulaire, ce qui lui permet de répondre à tous les besoins de l'utilisateur.

## Constitution d'une «entité» intelligente

Nous allons tenter ici de schématiser l'organisation d'un système intelligent, et ceci par l'assemblage de boîtes noires, nous entrerons dans les détails un peu plus loin.

### La mémoire

En nous reportant à la figure 1, nous remarquons que pour la partie «cerveau», un dialogue permanent doit pouvoir se faire entre mémoire «boîte aux lettres» et l'unité d'opérations et de séquencements ; or, toutes les informations emmagasinées dans la mémoire doivent être recueillies dans le temps le plus court possible afin d'être traitées en «temps réel». Pour arriver à cette finalité, il suffit d'imaginer que la mémoire est organisée en «étages». A chaque étage se trouve l'information soigneusement rangée qu'il suffit d'aller chercher à condition de savoir qu'elle se trouve là ! Une telle organisation est dite aléatoire parce qu'il n'y a qu'une information par étage et qu'il est ainsi facile de la

trouver. Par exemple (fig. 2), à l'étage 5 se trouve Jean, et Françoise à l'étage numéro 8...

Nous proscrivons d'emblée l'organisation séquentielle qui obligerait, pour aller chercher Françoise, de passer par les appartements de Catherine, Vincent, Claude..., quelle perte de temps et quels dérangements pour tout le monde ! (fig. 3).

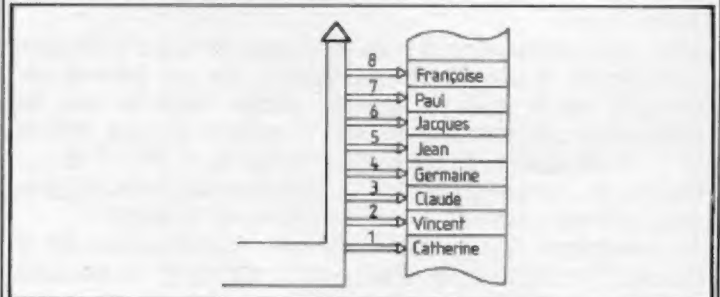


Fig. 2 : Organisation aléatoire de la mémoire.

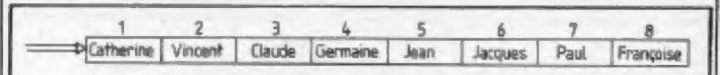


Fig. 3 : Organisation séquentielle de la mémoire.

Afin de bien comprendre ce qui suit, prenons l'exemple suivant :

Nous devons aller chercher Françoise pour aller au cinéma, elle habite le 8<sup>e</sup> étage d'un immeuble un peu vieillot du centre de Montpellier. L'escalier permettant d'accéder aux appartements de l'immeuble est très étroit ; aussi, le syndic des copropriétaires oblige-t-il les personnes à emprunter l'ascenseur pour monter, et l'escalier pour descendre (il est moins essoufflant de descendre un escalier que de le monter).

Une telle organisation permet une plus grande rapidité d'exécution car plusieurs personnes peuvent emprunter l'escalier et/ou l'ascenseur pour des directions différentes ; par contre, si l'ascenseur tombe en panne, nous imaginons sans peine la confusion que cela entraînerait si tout le monde se trouvait en même temps dans l'escalier ; Françoise mettra pas mal de temps pour rejoindre son appartement du 8<sup>e</sup> étage !...

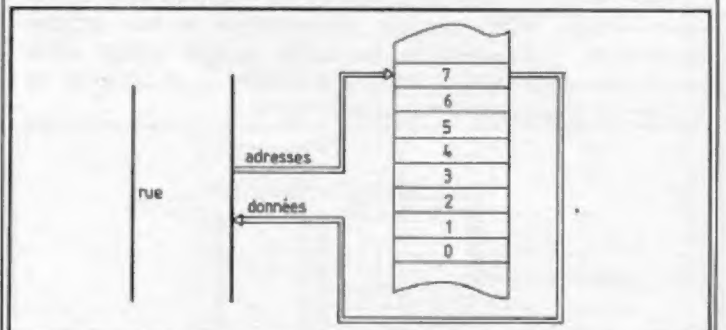


Fig. 4 : Structure mémoire standard (on commence toujours par 0).

Nous introduirons ici notre premier terme technique : L'escalier et l'ascenseur sont des lieux communs que tout le monde peut emprunter : on les appelle «BUS».

L'escalier qui permet d'arriver à l'étage s'appelle un Bus d'Adresses.

L'ascenseur qui récupère le «contenu de l'étage s'appelle Bus de Données.

Une telle architecture est appelée architecture standard : tous les micro-ordinateurs utilisent une telle architecture avec cependant une capacité plus ou moins étendue (l'escalier est plus ou moins large et l'ascenseur plus ou moins grand).

Ainsi, pour notre immeuble de 8 étages, le bus d'adresses comprendra 3 bits (3 fils de liaison), ce qui permet de compter de 0 à 7 (le premier étage étant le rez de chaussée), un bus de 16 fils comptera jusqu'à 65535 ( $2^{16} - 1$ ), un bus de 20 fils jusqu'à  $1048575 = 2^{20} - 1$ , etc... Le bus de données quant à lui, comprendra 8 fils, on dira ainsi qu'il est organisé en octets (Bytes en anglais).

En admettant que Françoise habite l'appartement 85 à l'étage 7, on trouvera la valeur binaire 10000101 à l'adresse 111. On remarque aussi qu'à chacune des adresses, on peut ranger toute valeur comprise entre 0 (00000000) et FF (11111111) soit  $0 \leq \text{Données} \leq 255$  décimal.

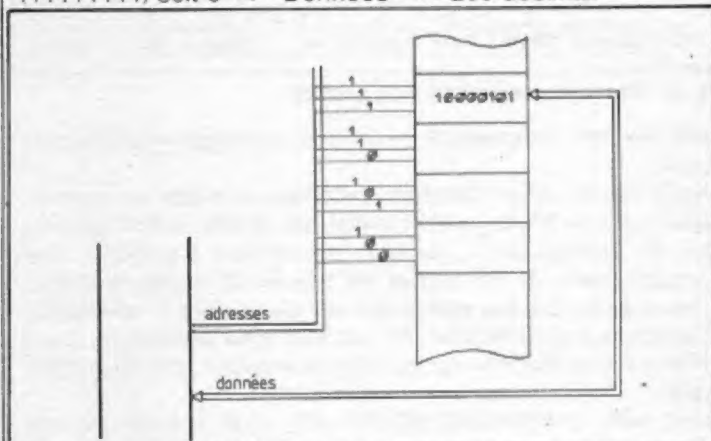


Fig. 5 : Organisation de la mémoire en octets : la capacité de la mémoire correspond à  $2^{\text{nombre de fils d'adresses}} \times 8 = 2^3 \times 8 = 64$  bits.

Il n'est pas bien difficile d'imaginer une structure mémoire organisée en mots (les données sont codées sur 16 bits) : imaginons un building style «Tour Montparnasse» ou «Tour Infernale» pour les ennemis de ce type d'architecture.

La cage d'escalier accède à plusieurs appartements sur le même étage, mais chaque appartement a son propre ascenseur. L'information recueillie à cet étage sera constituée de 2 valeurs dont la concaténation formera un mot de 16 bits (codé sur 2 octets) fig. 6.

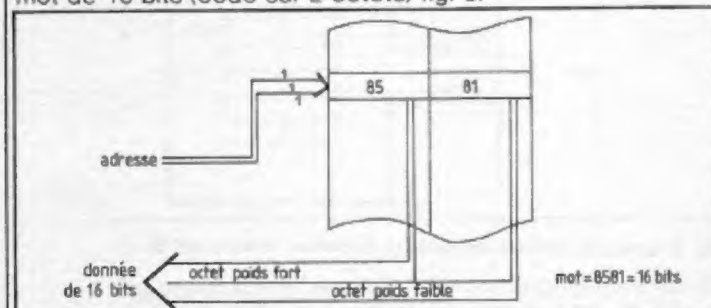


Fig. 6 : Organisation de la mémoire en mots : ici à l'adresse 7 on trouve le mot de 16 bits : 8581 recueilli sur le bus de données.

Il n'y a pas de limitation à l'organisation de la mémoire, elle peut aussi être étendue sur un long mot (32 bits), sur un très long mot (64 bits) etc...

Mais quel casse-tête pour le programmeur, aussi contentons-nous de l'octet !

Enfin, dernière remarque : l'appartement de Françoise pourrait très bien être occupé par Julien ; Françoise prêtant son appartement lorsqu'elle doit s'absenter quelques jours (attention, deux données ne peuvent coexister à la même adresse... il y a dans ce cas un conflit). Cela nécessite d'avoir un bus de données bi-directionnel.

Pour la donnée pouvant être rangée à l'adresse X, l'opération s'appellera une écriture (WRITE) ; pour la donnée pouvant être prise depuis l'adresse X, l'opération s'appellera une lecture (READ) fig. 7.

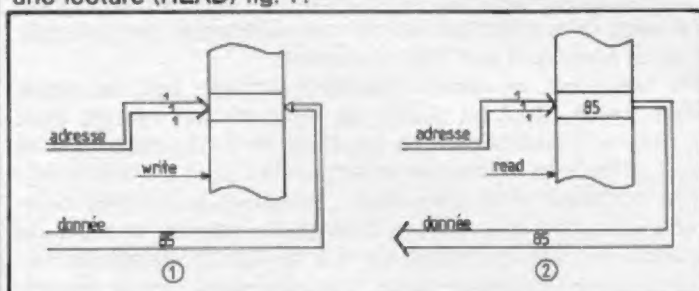


Fig. 7 : Le bus de données est bidirectionnel.

1. Opération d'écriture : le bus de donnée est orienté vers la mémoire grâce à l'ordre «Write».
2. Opération de lecture : le bus de donnée a changé de sens grâce à l'ordre «Read».

### L'unité centrale

Nous avons vu précédemment qu'une «unité centrale» était indispensable pour assurer les séquençements des opérations demandées dans un programme. Ces opérations peuvent être de type arithmétique ou logique, il sera judicieux d'intégrer une Unité Arithmétique et Logique (UAL) au sein de l'unité centrale.

Enfin, l'unité centrale ne nous donnera un résultat que dans le cas où elle disposera de tous les renseignements dont elle a besoin ; il faudra donc la munir d'une mémoire bloc-note (scratch pad en anglais), celle-ci étant aussi réduite que possible et se résume souvent à quelques registres. Prenons le rôle d'un concepteur de systèmes et construisons notre première machine intelligente que nous appellerons MOPET (Micro Ordinateur Pour Etudes Techniques)... et maintenant, laissons notre MOPET faire son show (il fallait bien la placer celle-là) fig. 8.

Avant toute chose, considérons que la mémoire est subdivisée arbitrairement par nous en deux zones : une zone programme qui contiendra notre programme, et une zone données qui contiendra les résultats des opérations demandées par notre programme.

Toute opération demandée à l'unité centrale s'écrit suivant une syntaxe particulière :

- une ligne de programme = (Code Opérateur) + (Opérande) = une instruction
- ou une ligne de programme = (Additionne) + (2 + 2) = une instruction

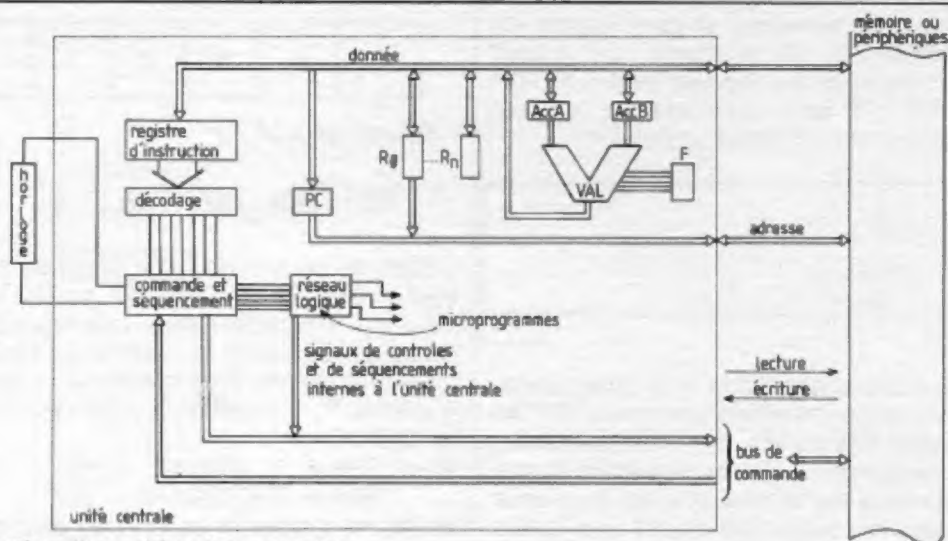


Fig. 8 : Constitution schématique d'une unité centrale -standard-.

- ou une ligne de programme = (Charge dans R0) + (3) = une instruction
- ou une ligne de programme = (Stocke à l'adresse 2000) + (la valeur 500) = une instruction
- etc...

Prenons un exemple :

Il s'agit d'additionner 2 avec 2 et de placer le résultat dans la zone données de la mémoire à l'adresse 2000 ; le programme situé à l'adresse 1000 s'écrit ainsi :

Adresse 1000      additionne 2 avec 2

Adresse 1000 + X      Stocke le résultat en 2000

Le séquençement sera le suivant :

- ① L'unité de séquençement a positionné la ligne de lecture à 0 (un 0 est toujours actif), permettant de lire dans la mémoire (on verra par la suite qu'une unité centrale passe la majorité de son temps à lire !). C'est le compteur de programme PC qui positionne le bus d'adresses à la valeur 1000, permettant ainsi de trouver la 1ère instruction (t1). Le code opératoire transite par le bus de données pour être décodé dans le registre d'instructions (t2).

puisse aller chercher l'opérande (t5) en mémoire programme, elle ouvre les accumulateurs (t6) de façon qu'ils admettent l'opérande se trouvant enfin sur le bus de données.

Le résultat de l'addition t8 se trouve automatiquement stocké dans l'un des accumulateurs (supposons par convention que ce soit A).

Nous remarquons que l'instruction d'addition s'exécute en 8 séquences. Si une horloge régit chaque séquence, on en déduit que cette instruction s'est exécutée en 8 périodes.

- ③ L'unité de séquençement a positionné le PC à l'adresse suivante, qui correspond à l'instruction suivante (t1), le code opératoire transite par le bus de données pour être décodé (t2). Puisqu'il s'agit d'une instruction de stockage à une adresse définie (ici 2000), l'unité de séquençement positionne AccA en sortie et aiguille le bus de données sur un registre compteur de programme temporaire (non adressable par le programmeur) (t3).
- ④ Le PC s'incrmente d'un pas pour prendre l'opérande qui est chargé immédiatement dans le PC temporaire (t4).

est chargé immédiatement dans le PC temporaire (t4).

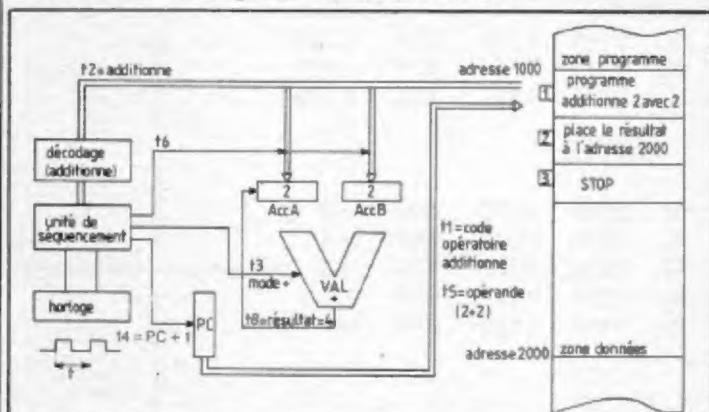


Fig. 9 : Le MOPET additionne.

- ② L'unité de séquençement positionne l'UAL en mode addition (t3) puis fait avancer le PC d'un pas (t4) pour qu'il

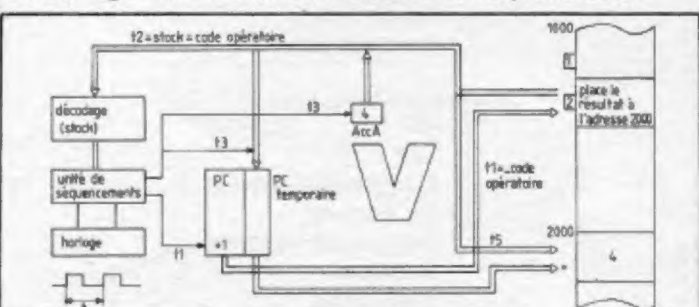


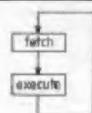
Fig. 10 : Le MOPET en stockage.

La donnée transite ensuite par le bus de donnée pour être stockée à l'adresse pointée par le PC temporaire (t5). Le travail est fini en 5 périodes d'horloge.

Il a fallu en tout 13 périodes d'horloge pour exécuter une modeste opération d'addition... un meilleur concepteur aurait sûrement fait beaucoup mieux.



Si vous ne l'avez pas encore remarqué, on peut résumer les opérations précédentes en 2 moments ou cycles continuellement recommencés, il s'agit des cycles **FETCH** (pour chercher) et **EXECUTE** (pour exécuter) que l'on peut résumer par l'organigramme ci-dessous :



L'unité centrale, en cours d'exécution d'un programme, passe son temps à chercher une instruction (cycle **FETCH**) puis à l'exécuter (cycle **EXECUTE**). Son intelligence (en l'occurrence son microprogramme interne) ne lui permet pas de savoir qu'un programme est terminé et c'est donc pour cette raison qu'on place une instruction d'arrêt pour lui dire «c'est fini», autrement l'unité centrale continuerait indéfiniment l'exécution d'un programme imaginaire faisant suite aux instructions exécutables. On dira alors que le système est «planté» (pour utiliser le jargon cher aux informaticiens !).

Nous allons maintenant élaborer la partie matérielle de notre MOPET, commençons donc par construire l'Unité Arithmétique et Logique (UAL).

## L'UAL du MOPET

Comme son nom l'indique, cette unité doit être capable d'effectuer des opérations Arithmétiques et Logiques ; les plus simples sont : (tableau 1) :

Opérations arithmétiques :  
 Multiplication par 2  
 Division par 2  
 Complémentation à 2  
 Addition  
 Soustraction  
 Comparaisons

Opérations logiques :  
 ET  
 OU  
 Complémentation (à 1)  
 Ou Exclusif  
 Décalages logiques  
 Rotations  
 Test de bit

Ces opérations s'effectueront sur un mot de 8 bits appelé octet, contenu dans l'un des accumulateurs. Chaque bit de cet octet est affecté d'un poids, par exemple le bit numéro 7 est affecté du poids  $2^7 = 128$ , le bit numéro 0 est affecté du poids  $2^0 = 1$  etc...

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

$$B7 = 2^7 \quad b6 = 2^6 \quad 2^5 \quad 2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0$$

$$128 \quad 64 \quad 32 \quad 16 \quad 8 \quad 4 \quad 2 \quad 1$$

L'octet 85 sera codé **1000 0101** en binaire et sera donc égal à :  $2^7 + 2^2 + 2^0 = 128 + 4 + 1 = 133$  en décimal.

Pour s'y reconnaître dans ces équivalences, on prendra l'habitude de coder le nombre en base hexadécimale en utilisant le préfixe \$ ; le nombre en base binaire en utilisant le préfixe % ; le nombre en base décimale en n'utilisant pas de préfixe.

$$\text{Ainsi : } \$85 = \% 10000101 = 133$$

$$\$66 = \% 01100110 = 102$$

En fait, le problème est un peu plus complexe car il faut pouvoir représenter les nombres en valeurs signées, c'est-à-dire qu'il faudra tenir compte des valeurs négatives ; par convention, on définira le bit n° 7 comme un bit de signe, ainsi si  $b7 = 0$  le nombre codé sur 8 bits est **POSITIF** ; si  $b7 = 1$ , il est **NEGATIF**.

Notre nombre N pourra donc être compris entre deux extrêmes qui sont :

$$\% 01111111 = \$7F = 2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 = 2^7$$

$$-1 = 127 \text{ et } \% 10000000 = \$80 = -128 = -2^7$$

On trouvera facilement ce nombre négatif en complément à 1 le nombre positif et en ajoutant 1 au résultat ; ainsi, si N = une valeur positive, N en complément à 2 sera égal à  $N2's = N + 1$ .

Ex. si N = \$00  $N2's = \$FF + \$01 = \$00$  (ce qui est normal)

si N = \$01  $N2's = \$FE + \$01 = \$FF$  (1 et -1 en décimal)

si N = \$10  $N2's = \$EF + \$01 = \$11$  (16 et -16 en décimal)

On en déduit facilement le tableau 2 suivant :

%	0111	1111	\$7F	+127
%	0111	1110	\$7E	+126
%	0111	1101	\$7D	+125

%	0000	0010	\$02	+2
%	0000	0001	\$01	+1
%	0000	0000	\$00	+0
%	1111	1111	\$FF	-1
%	1111	1110	\$FE	-2

%	1000	0010	\$82	-126
%	1000	0001	\$81	-127
%	1000	0000	\$80	-128

+

-

Notre MOPET travaillera donc en permanence en complément à 2, il tiendra compte de ce fait dans toutes les opérations arithmétiques qu'on lui demandera d'effectuer et l'ignorera lors d'opérations logiques.

Fort de cet enseignement, nous sommes dorénavant en mesure d'élaborer une Unité Arithmétique et Logique !

## Fonctions logiques

Compte tenu du tableau 1, nous devons réaliser des opérations logiques booléennes classiques. L'élément de base sera le transistor unipolaire puisque la technologie utilisée sera du type MOS (figures 11 et 12).

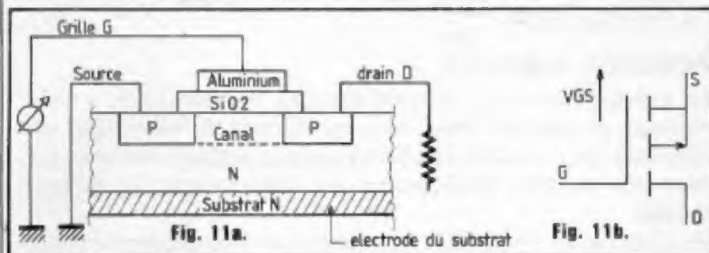


Fig. 11 : Transistor du type PMOS.

Sur un substrat de type N on développe deux îlots de type P, l'un appelé Source et l'autre Drain. La grille est séparée du substrat par une couche d'isolant qui est de l'oxyde de silicium.

Une tension négative de grille provoque un afflux de porteurs positifs (trous) dans l'espace source-drain (fig. 11a).

Puisque le canal ainsi constitué est riche en trous, on dira que le transistor est du type P, d'où l'appellation PMOS (fig. 11b).

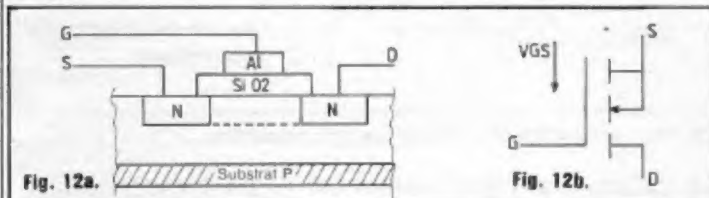


Fig. 12 : Transistor du type NMOS.

En développant deux îlots de type P sur un substrat de type N, on réalise un transistor NMOS (fig. 12a).

Une tension positive de grille provoquera le passage d'électrons majoritaires dans l'espace source-drain. Ces deux porteurs étant négatifs, le canal formé sera donc du type N d'où l'appellation NMOS (fig. 12b).

Il est aisé de réaliser un inverseur en disposant deux transistors PMOS et NMOS tête bêche (fig. 13).

Ainsi, si  $E = 0$ , la tension  $V_{GS}$  du NMOS est égale à 0, il est bloqué et offre donc une résistance infinie entre drain et source ; par contre, le PMOS se sature puisque sa tension de grille est plus faible que sa tension de source, on trouve ainsi la tension  $V_{dd}$  sur la sortie  $S$ .

Le NMOS va se saturer si  $E = 1$ , d'où  $S = 0$ .

On réalisera une fonction NAND sans problème en disposant 2 NMOS en cascade : pour vérifier la fonction, il faudra bien que le premier et le 2<sup>e</sup> transistor conduisent.

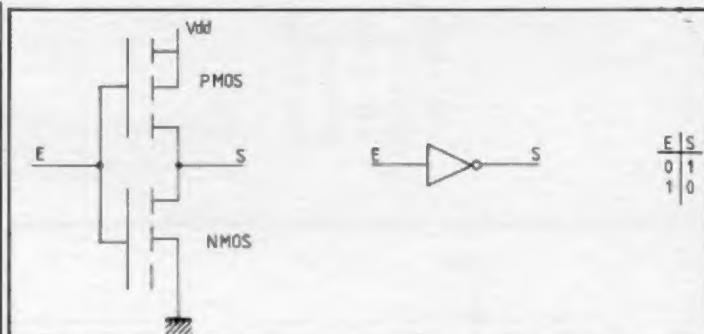


Fig. 13 : Un inverseur.

Remarquons la présence d'une résistance de polarisation qui n'est autre qu'un transistor MOS dont la grille est reliée à la source (fig. 14).

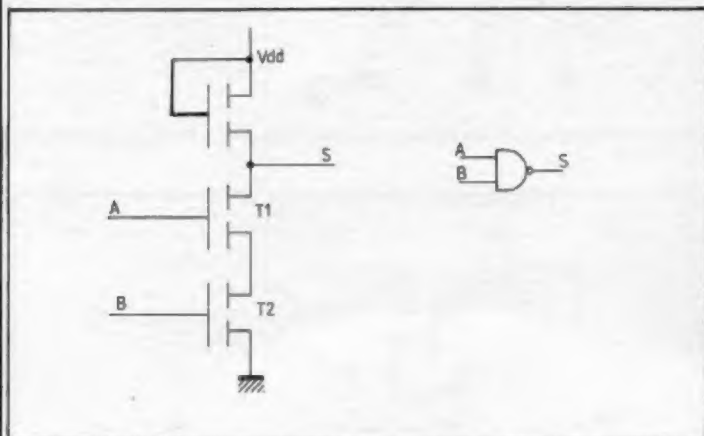


Fig. 14 : Un NAND.

Un NOR sera réalisé par deux transistors disposés en parallèle (fig. 15). Les fonctions ET, OU seront simples à réaliser à partir des éléments déjà connus (fig. 16a et b).

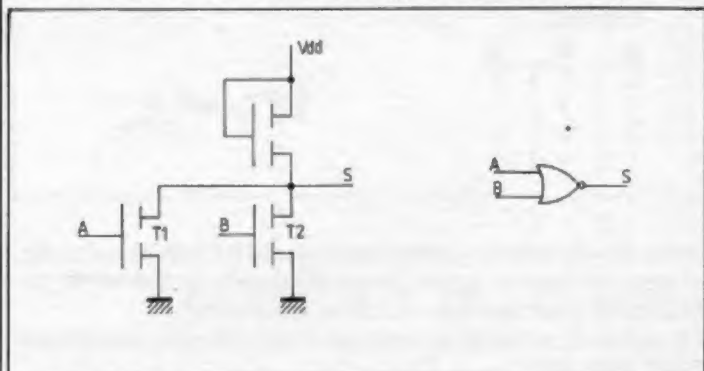


Fig. 15 : Un NOR.

A	B	S
0	1	0
1	0	0
0	0	1
1	1	0

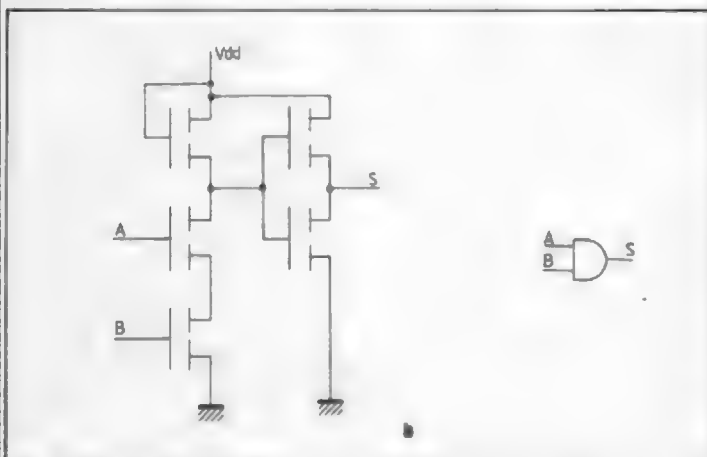
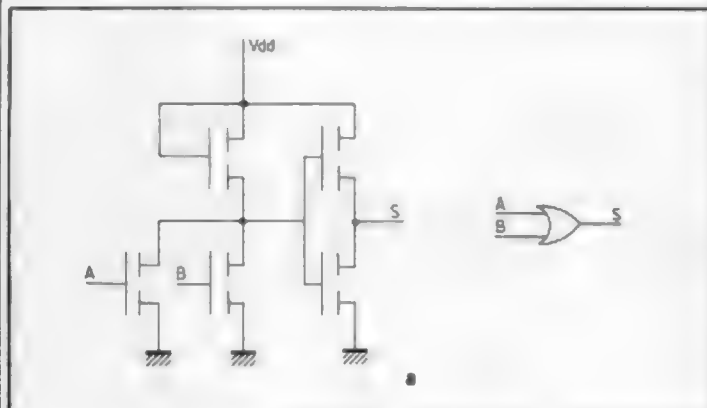


Fig. 18 : (a) un OR (b) un AND.

La dernière fonction combinatoire classique est la fonction OU exclusive (Exclusive OR = EOR), elle est déduite de la table de vérité suivante :

Entrées			
A	B	S	
0	0	0	
0	1	1	
1	0	1	
1	1	0	

$$S = A\bar{B} + \bar{A}B = A \oplus B$$

Cette fonction sera réalisée simplement à l'aide des circuits ci-dessus. Pour y arriver, nous utiliserons le théorème de MORGAN pour qui  $A.B = \overline{\overline{A} + \overline{B}}$  et  $\overline{A+B} = \bar{A}.\bar{B}$ . On arrive au schéma suivant qui n'utilisera que des portes NAND (fig. 17) :

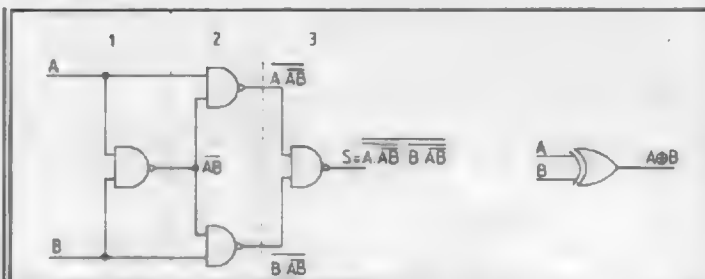


Fig. 17.

$$\begin{aligned} S &= A.\bar{A}B + \bar{A}B.AB = A.AB + B.AB = A.AB + B.AB \\ &= \bar{A}B(A+B) = (\bar{A} + \bar{B})(A+B) = \bar{A}B + \bar{B}A = A \oplus B \\ &= \text{Fonction EOR} \end{aligned}$$

### Fonction mémoire

La logique décrite ci-dessus est dite combinatoire, il nous manque un élément important qui permet de mémoriser les résultats des opérations combinatoires, il s'agit de l'élément mémoire obtenu simplement en introduisant le facteur temps.

En effet, si en un temps  $t$ , on mémorise une information que l'on pourra récupérer à l'instant  $t+1$  sans aucune modification, on aura certes perdu l'aspect fugitif de cette information, mais on aura gagné une nouvelle possibilité qui permet de mémoriser indéfiniment une information.

Un tel élément capable de stocker une information binaire s'appelle une mémoire ou bascule ou flip flop ou latch.

La figure 18a représente une bascule avec des portes NOR.

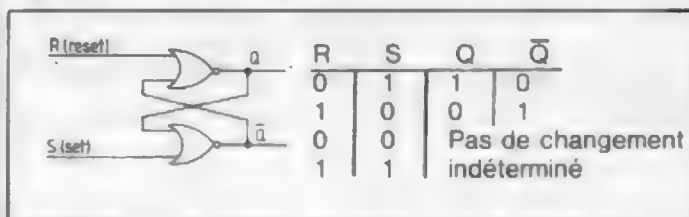


Fig. 18a : Une bascule RS n'accepte que trois possibilités.

Le malheur d'une telle bascule est qu'elle accepte un état indéterminé ( $R=S=1$ ) que l'on pourra supprimer en obligeant R et S à avoir deux états complémentaires, il s'agit de la bascule D (fig. 18b).

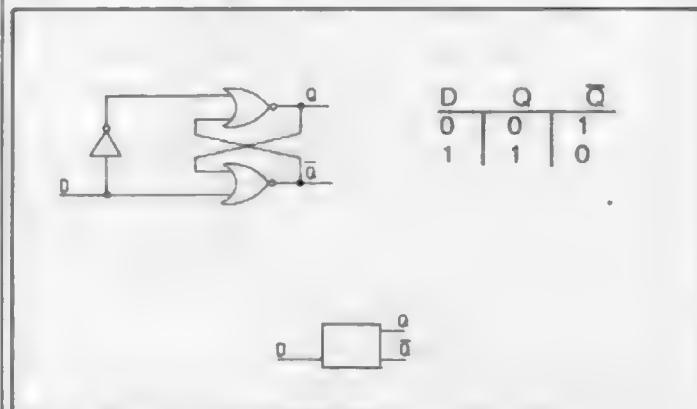
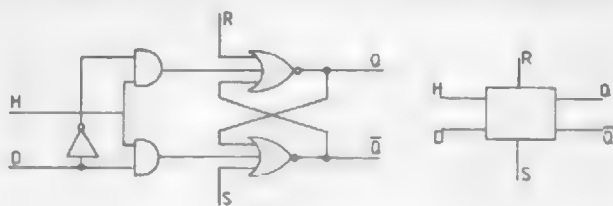


Fig. 18b : Bascule type D.





R	S	H	D	Q	Q̄
1	X	X	X	0	X
X	1	X	X	X	0
1	0	X	X	0	1
0	1	X	X	1	0
0	0	1	0	0	1
0	0	1	1	1	0

Fig. 18c : Bascule D synchronisée sur une horloge avec SET et RESET.

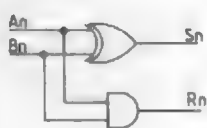
Pour rester synchrone sur des événements extérieurs, il suffira d'ajouter une entrée horloge, de même qu'une entrée Set (S) initialisera la sortie Q de la bascule à 1 et Reset (R) à 0.

C'est une bascule de ce type qui constituera les registres et accumulateurs de notre MOPET (fig. 18c).

## Fonction arithmétique

### L'addition

Une première opération à réaliser est l'addition ; en raisonnant sur les bits de rang n de 2 valeurs A et B, on obtient le tableau de la figure 19.



An	Bn	Sn	Rn
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Fig. 19 : Un demi-additionneur.

Rn étant la retenue, on obtient deux équations :

$$S_n = A_n \bar{B}_n + \bar{A}_n B_n = A_n \oplus B_n$$

$$R_n = A_n B_n$$

L'additionneur étudié est encore incomplet puisqu'il faut tout de même tenir compte de la retenue provenant de l'addition sur les bits de poids n-1.

On arrive ainsi au tableau de la figure 20.



An	Bn	Rn-1	Sn	Carry Rn
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S_n = \bar{A}_n \bar{B}_n R_{n-1} + \bar{A}_n B_n \bar{R}_{n-1} + A_n \bar{B}_n \bar{R}_{n-1} + A_n B_n R_{n-1}$$

$$R_n = A_n B_n \bar{R}_{n-1} + A_n \bar{B}_n R_{n-1} + \bar{A}_n B_n R_{n-1} + A_n B_n R_{n-1}$$

$$S_n = R_{n-1} (A_n B_n + \bar{A}_n B_n) + \bar{R}_{n-1} (A_n \bar{B}_n + B_n \bar{A}_n)$$

$$S_n = R_{n-1} (A_n \oplus B_n) + \bar{R}_{n-1} (A_n \oplus B_n)$$

$$S_n = R_{n-1} \oplus (A_n \oplus B_n)$$

$$R_n = A_n B_n (R_{n-1} + \bar{R}_{n-1}) + R_{n-1} (\bar{A}_n B_n + A_n \bar{B}_n)$$

$$R_n = A_n B_n + R_{n-1} (A_n \oplus B_n) \quad \text{Fig. 20 : Additionneur binaire de poids N.}$$

Un additionneur complet sera donc formé d'une chaîne d'étages additionneurs identiques à celui de la figure 20.

Sur la figure 21, on voit que la ligne de départ à 1 donnera les informations A1, B1, A2, B2... An, Bn simultanément aux additionneurs.

Compte tenu du délai d'addition, la ligne résultat qui passera à 1 permettra d'enregistrer la somme S0, S1, S2... Sn dans une mémoire.

On se réservera aussi la possibilité d'assurer l'addition en tenant compte d'une carry éventuelle (appelée ici Cy) provenant d'une addition précédente (fig. 21).

En effet, si nous faisons :

$$\begin{array}{r} \$80 \\ + \$80 \\ \hline \text{C} \end{array} \quad \begin{array}{r} 1000 \quad 0000 \\ + 1000 \quad 0000 \\ \hline \text{C} \end{array}$$

$$\text{C} \quad \begin{array}{r} 1 \\ \hline \end{array} \quad \begin{array}{r} \$00 = R \end{array}$$

$$\text{C} \quad \begin{array}{r} 1 \\ \hline \end{array} \quad \begin{array}{r} 0000 \quad 0000 = R \end{array}$$

Nous obtenons en réalité la valeur \$100 qui sera codée sur 9 bits ! Comme cela n'est pas possible, nous nous contenterons d'un résultat sur 8 bits en mémorisant toutefois la carry qui nous sera bien utile si nous codons le résultat sur 16 bits.

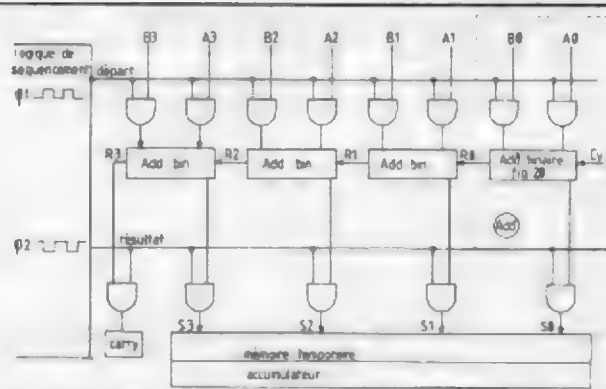


Fig. 21 : Un additionneur complet.

## La soustraction

A ce stade, nous avons deux solutions :

1) Ecrire l'équation booléenne de la soustraction et en déduire le schéma de principe qui sera forcément différent de celui d'un additionneur.

2) Garder la structure de l'additionneur en ajoutant l'élément soustraction. C'est bien entendu la seconde solution qui sera choisie puisqu'elle part d'un principe fort simple : pour soustraire, on va additionner !

Nous avons vu qu'un nombre peut être représenté en valeurs négatives en utilisant le complément à 2 : la plage de variation d'une valeur binaire codée sur 8 bits sera comprise entre +127 et -128 d'où  $A-B = A + (\bar{B} + 1)$ , on conserve ainsi l'additionneur binaire comme le montre clairement la figure 22.

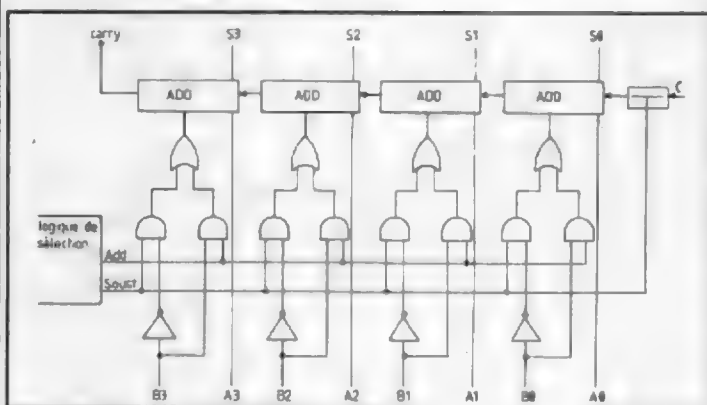


Fig. 22 : L'additionneur-soustracteur.

Si Add = 1 → Soust = 0 on obtient  $S_i = A_i + B_i$  (la carry C éventuelle est ajoutée sur l'élément ADD de poids le plus faible).

Si Add = 0 → Soust = 1 on obtient  $S_i = A_i + (B_i + 1)$  la valeur 1 est ajoutée au premier élément ADD.

Chaque élément ADD représente l'intérieur de l'espace entre pointillés de la figure 21.

La possibilité Add = Soust = 1 est impossible à exécuter grâce à la logique de sélection, par contre Add = Soust = 0 inhibe l'additionneur soustracteur.

## La comparaison

L'opération de comparaison est extrêmement importante pour qui veut exécuter un programme différent suivant le résultat de la comparaison. Supposons que l'on veuille comparer 2 nombres A et B. Il se peut que :

$A = B$  ou  $A - B = 0$   
 $A > B$  ou  $A - B > 0$   
 $A < B$  ou  $A - B < 0$   
 $A \leq B$  ou  $A - B \leq 0$

On peut aussi comparer un nombre avec lui-même ainsi :

$A = 0; A > 0; A > = 0; A < 0; A < = 0$

Une comparaison est donc équivalente à une soustraction entre deux nombres, mais le résultat de cette soustraction ne servira qu'à positionner des bits particuliers nommés «flags» ou «drapeaux» qu'il suffira de tester si besoin est. Les flags seront positionnés à 1 si la condition reste vraie (TRUE), ils seront à 0 si la condition est fausse (FALSE).

On voit ci-dessous que les flags importants sont :

- Le flag N (négatif) : si le résultat de la comparaison est négatif, on aura  $N = 1$ , s'il est positif on aura  $N = 0$

- Le flag Z (Zero) : si le résultat de la comparaison est nul, on aura  $Z = 1$ , s'il est différent de 0 on aura  $Z = 0$

- Le flag V (Overflow), si le résultat de la comparaison dépasse la capacité de 8 bits  $V = 1$  (donc si le résultat est  $< -128$  ou si le résultat est  $> +127$ ) autrement  $V = 0$

Pour bien comprendre ce qui se passe dans ce dernier cas, faisons :

$A = -123$  et  $B = +15$   
 $A - B = -138 < -128$  (limite des 8 bits)  
 $A = 1000\ 0101$  représente  $-123$   
 $-B = 1111\ 0001$  représente  $-15$  ( $-B = \bar{B} + 1$ )

$0\ 1\ 1011\ 0110 =$  une valeur positive ( $+118$ ) alors qu'elle devrait être négative.

V sera donc le ou exclusif de N et C →  $V = N \oplus C$  (entraînez-vous avec d'autres exemples).

Nous avons tous les éléments en main pour réaliser l'opération de comparaison. Construisons le tableau ci-dessous :

A	B	$A = B$ $A - B = 0$	$A > B$ $A - B > 0$	$A \geq B$ $A - B \geq 0$	$A < B$ $A - B < 0$	$A \leq B$ $A - B \leq 0$	N	Z
0	0	1	0	1	0	1	0	1
0	1	0	0	0	1	1	1	0
1	0	0	1	1	0	0	0	0
1	1	1	0	1	0	1	0	1

Pour  $A = B$  on a  $\bar{A}\bar{B} + AB = \bar{A} \oplus B = S$

Pour  $A > B$  on a  $\bar{A}\bar{B} = S1$

Pour  $A < B$  on a  $\bar{A}\bar{B} = S2$

Pour  $A \leq B$  on a  $\bar{A}\bar{B} + AB + \bar{A}\bar{B} = S + S2$

Pour  $A \geq B$  on a  $\bar{A}\bar{B} + AB + \bar{A}\bar{B} = S + S1$

La condition  $Z=1$  est donnée par S qui représente l'équation d'un 1/2 additionneur complémenté à 1... et  $N=1$  pour  $S2$  mais en fait N n'est rien d'autre que la copie du bit de poids le plus fort ! Ce qui nous évitera un schéma trop compliqué (fig. 23).

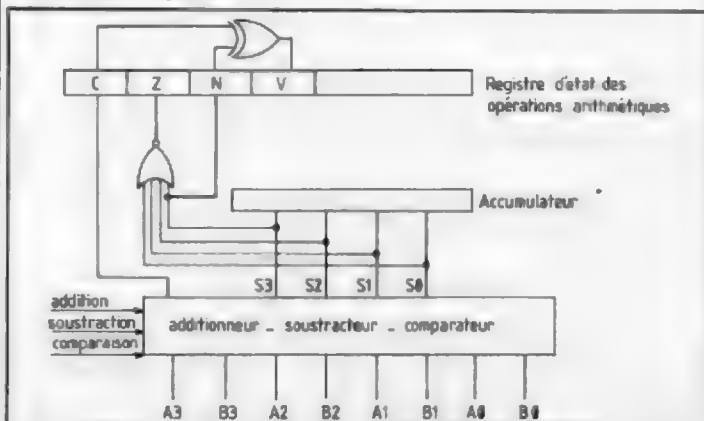


Fig. 23 : Une comparaison des 2 mots linéaires positionne les flags Z, N et V.

## Les décalages

Pour ne pas compliquer le problème outre mesure, il serait possible de réaliser un multiplicateur et un diviseur 8 bits par 8 bits avec le résultat sur 16 bits dans le 1er cas, et sur 8 bits dans le second cas + 8 bits de reste.

Mais il est cependant plus simple d'aboutir à un résultat identique à une multiplication et à une division en effectuant de simples décalages. En effet, chaque élément binaire d'un nombre codé sur 8 bits représente un poids qui est un multiple de 2.

En lisant les bits de la droite vers la gauche, on trouve :

bit 0 =  $b_0 = 2^0 = 1$

bit 1 =  $b_1 = 2^1 = 2$

bit 2 =  $b_2 = 2^2 = 2^1 \times 2^1$

bit 3 =  $b_3 = 2^3 = 2^1 \times 2^1 \times 2^1$  etc...

Donc, chaque déplacement sur la gauche représente une puissance de 2 (ce qui fait qu'on multiplie chaque fois par 2) de même, chaque déplacement sur la droite représente une division par 2.

Par exemple : 12 en décimal donne

0	0	0	0	1	1	0	0
128	64	32	16	8	4	2	1

En faisant un décalage à gauche, on multiplie bien par 2 puisqu'on obtient :

C	V	N	0	0	0	0	0	0	1	1	0	0	0
0	0	0											

ce qui donne bien 24 en décimal.

On remarquera qu'un tel décalage impose un zéro introduit sur le bit de poids le plus faible alors que le bit de poids fort est introduit dans la carry, puisque  $b_7 = 0$ , on obtient  $N = 0$  d'où  $V = N \oplus C = 0$ .

Si on prend maintenant un nombre négatif (-12 par exemple) qui s'écrit :  $C \leftarrow 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \leftarrow 0$ , un décalage à gauche donnera

V N C  
0 1 1 1 1 1 0 1 0 0 0 0 qui donne bien -24... si  $V = 1$  cela voudrait dire qu'il y a dépassement de capacité (résultat < -128).

On pourrait penser qu'un décalage à droite décalqué sur le même principe que le décalage à gauche pourrait suffire pour la division par 2 ; mais c'est oublier la présence du bit de signe qui impose de le dupliquer après chaque décalage afin d'obtenir un résultat correct.

Ainsi +12 donne 00001100 après un décalage à droite, on obtient :

0 0 0 0 0 1 1 0 → C N ce qui donne bien 6 (V

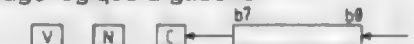
n'aura pas lieu d'être positionné pour une division !).

Mais -12 donne 11111010 après un décalage à droite on obtient :

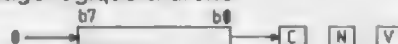
1 1 1 1 1 0 1 0 → C N ce qui donne bien -6.

Ces deux fonctions de décalage sont du type arithmétique. On peut aisément introduire des fonctions logiques en les définissant ainsi :

- décalage logique à gauche



- décalage logique à droite



- rotation à gauche



- rotation à droite



On remarquera que le décalage logique à gauche est identique au décalage arithmétique à gauche, mais par contre le décalage arithmétique à droite est différent du décalage logique qui introduit un zéro sur le bit de poids fort (c'est donc pour cette raison que l'on aura toujours  $N = 0$ ).

Le but d'un décalage logique est de faire «tomber» un bit dans la carry, qu'il suffira ensuite de tester pour voir si ce bit est à 1 ou 0 (il existera bien sûr une instruction spécifique testant spécialement cette carry).

Un décalage logique change le mot contenu dans le registre, par contre, la rotation permet de récupérer ce mot après 8 tours.

Nous avons maintenant assez d'éléments pour construire le dernier élément de notre unité arithmétique et logique.

## Le registre (appelé encore accumulateur)

On appelle registre une mémoire linéaire à accès parallèle. Chaque case du registre est constituée d'une bascule accessible séparément, mais il est aussi possible de déplacer pas-à-pas l'information dans ce registre, de même qu'il est facile de l'utiliser en entrée-sorties parallèles.

La figure 24 donne un exemple de registre à chargement parallèle et décalage à droite.

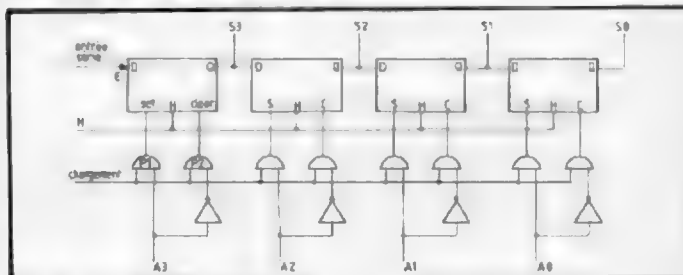


Fig. 24 : Un registre parallèle-série.

• Chargement parallèle : un ordre de chargement permet de recopier le mot présent en  $A_3A_2A_1A_0$  sur  $S_3S_2S_1S_0$ . si  $A_3 = 1$ , la porte P1 est ouverte, d'où  $set = 1$  et  $Q = 1$  si  $A_3 = 0$ , la porte P2 est ouverte, d'où  $clear = 1$  et  $Q = 0$ . Dans ces 2 cas, on doit avoir la condition chargement = 1.

Décalage à droite : une impulsion sur la borne H des bascules type D permet de copier l'état présent sur l'entrée E vers l'entrée suivante F. Donc, 4 impulsions d'horloge auront transporté l'information E sur la sortie S0.

Une électronique supplémentaire permettrait d'assurer un décalage à gauche et une rotation dans les deux sens mais notre but n'est pas de trop compliquer un problème qui l'est déjà assez !

Une toute dernière remarque qui a son importance : les bascules en technologie MOS ne peuvent conserver



indéfiniment une information sans rafraîchissement interne. En effet, une information binaire est représentée par un « 1 » logique sur la grille d'un transistor, or, la capacité interne d'un tel transistor est relativement faible, ce qui fait qu'elle se décharge assez rapidement et on va ainsi perdre définitivement l'information que nous voulions mémoriser.

Il est donc important de prévoir une horloge de rafraîchissement qui aura pour but de recharger cette capacité afin de conserver cette information si importante. Les registres, qui peuvent être nombreux dans l'unité centrale, doivent pouvoir être déconnectés, il s'agit d'un 3<sup>e</sup> état qui introduit un nouveau signal de contrôle appelé «three state control» (TSC).

Cette fonction est très intéressante lorsque l'on branche plusieurs registres en parallèle : il suffit de placer en 3<sup>e</sup> état tous les registres non utilisés.

La figure 25 donne un exemple d'une porte à 3 états : la fonction 3 états est réalisée par une porte de transfert CMOS branchée en série avec la sortie.

Suivant la valeur de TSC, les deux transistors sont conducteurs et on a  $S = AB...$  ou les deux transistors sont bloqués et la porte NAND est en 3<sup>e</sup> état.

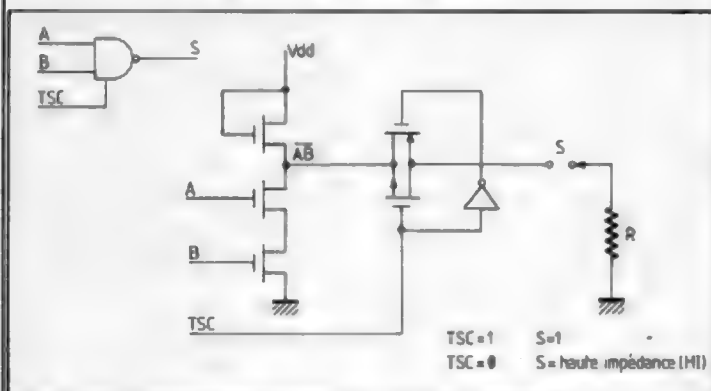


Fig. 25 : Une porte NAND avec borne TSC.

De nombreux buffers (type 74LS640-74LS645...) sont réalisés sur ce principe et on trouve, comme nous l'avons dit, l'équivalent dans notre MOPET et dans toutes les unités centrales.

### Réalisation d'une ALU complète

Une ALU complète simplifiée (celle de notre MOPET) est présentée figure 26a, par contre la figure 26b montre le schéma block d'une ALU classique à deux entrées : les buffers sont des registres temporaires normalement inaccessibles par le programmeur et le fonctionnement de cette ALU est présenté avec un accumulateur. On pourrait prévoir un deuxième accumulateur présent sur l'entrée B qui offrirait la possibilité de travailler avec des mots de 16 bits au lieu de 8.

### L'unité de commande du MOPET

Nous avons vu précédemment que tout ordre envoyé à l'unité centrale se traduit par deux mots binaires qui sont :

- le code opératoire COP
- l'opérande OP

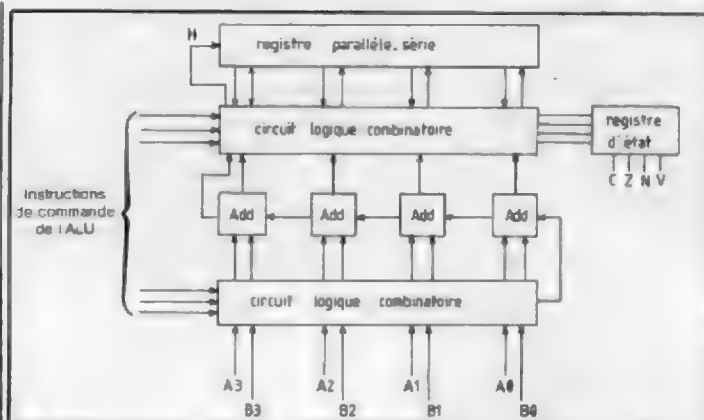


Fig. 26a : Une VAL simplifiée pouvant réaliser toutes les opérations logiques et arithmétiques.

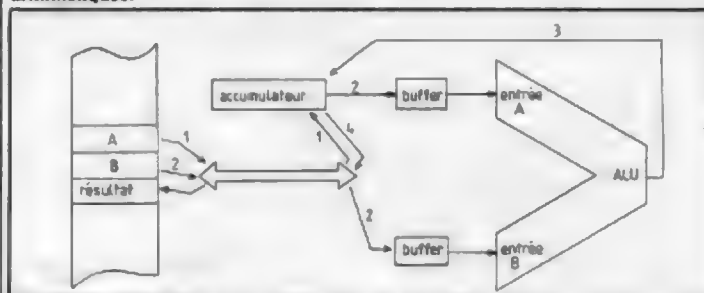


Fig. 26 b : Organisation des étapes pour une opération donnée.

- 1 A → Accumulateur
  - 2 B → Buffer → entrée B alu
  - 3 Résultat à la sortie de l'Alu dans l'accumulateur.
  - 4 Résultat Accu → mémoire.
- Accu → Buffer → entrée A alu

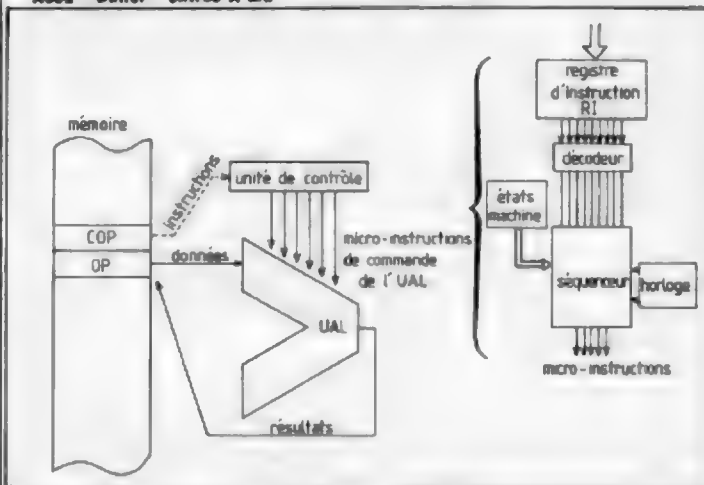


Fig. 27 : Rôle de l'unité de contrôle.

Nous verrons plus tard que l'opérande n'est, dans certains cas, pas nécessaire, de même que l'association COP + OP peut représenter plus de deux octets.

Le code opératoire représentera le type d'opération que l'UAL devra effectuer, tandis que l'opérande sera la donnée avec laquelle l'UAL devra travailler (rappelons que l'UAL a pour but essentiel, le traitement, la manipulation et la gestion des données : elle peut réaliser des fonctions logiques, arithmétiques, des comparaisons et des tests).

Si ce code opératoire est codé sur 8 bits, on a donc 256 types différents d'opérations à effectuer (de 00 à FF), ce qui

est intéressant mais le sera encore plus si le codage se fait sur 16 bits car on disposera alors de 65536 possibilités différentes !

La figure 27 montre le rôle joué par cette unité de contrôle. Une «loupe» disposée au-dessus de cette unité nous permet d'aboutir au schéma de la figure 8 :

L'unité de contrôle est constituée :

- d'un registre d'instructions RI qui stocke les instructions issues de l'emplacement mémoire adressé par l'unité centrale (son compteur de programme PC). Le RI n'est rien d'autre qu'un latch.
- d'un décodeur nécessaire au décodage du mot contenu dans le RI.
- d'un séquenceur central qui génère des micro-instructions indispensables au bon cheminement des informations. Notons que le séquenceur est du type synchrone, c'est-à-dire que les micro-instructions sont émises au rythme de l'horloge.

En résumé, le séquenceur central est l'organe qui génère les micro-instructions nécessaires au bon cheminement des informations.

## Décodage de l'instruction

Nous avons vu que le compteur de programme pointe sur le code opératoire disposé en mémoire.

Le code opératoire représentera une instruction particulière, par exemple un chargement dans l'accumulateur A d'une valeur \$3F.

La valeur \$3F étant l'opérande, se trouvera à l'adresse suivante c'est-à-dire à PC + 1.

Ce qui peut s'écrire : LDA # \$3F en langage assembleur ou 86.3F en hexadécimal.

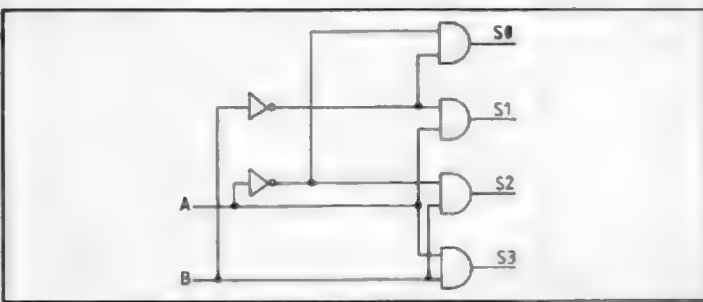
Laissons tomber pour l'instant le langage assembleur et examinons les valeurs hexadécimales.

Le code opératoire est égal à 86 et l'opérande = 3F.

## Décodeur 1 parmi N

La valeur 86 est mise dans le registre d'instructions RI puis est décodée à l'aide d'un décodeur digital à l'entrée duquel on applique le COP de 8 bits. La sortie sera donc constituée de  $N = 2^8 = 256$  fils.

Une seule sortie se trouve à 1 pour chacune des valeurs du code d'entrée (fig. 28).



entrées		sorties			
A	B	S0	S1	S2	S3
0	0	1	0	0	0
1	0	0	1	0	0
0	1	0	0	1	0
1	1	0	0	0	1

$$\begin{aligned} S0 &= \bar{A}\bar{B} \\ S1 &= A\bar{B} \\ S2 &= \bar{A}B \\ S3 &= AB \end{aligned}$$

Fig. 28 : Exemple d'un décodeur 1 parmi 4.

Le problème, c'est qu'on a à décoder toutes les valeurs comprises entre 00 et FF d'où une logique plutôt lourde et 256 fils en sortie prennent beaucoup de place sur une puce de silicium de 5 mm x 5 mm !

## Décodeur à mémoire morte

On va donc tâcher de diminuer la dimension de ce décodeur en faisant preuve d'astuces.

Par exemple, une matrice de diodes peut représenter un excellent décodeur (fig. 29).

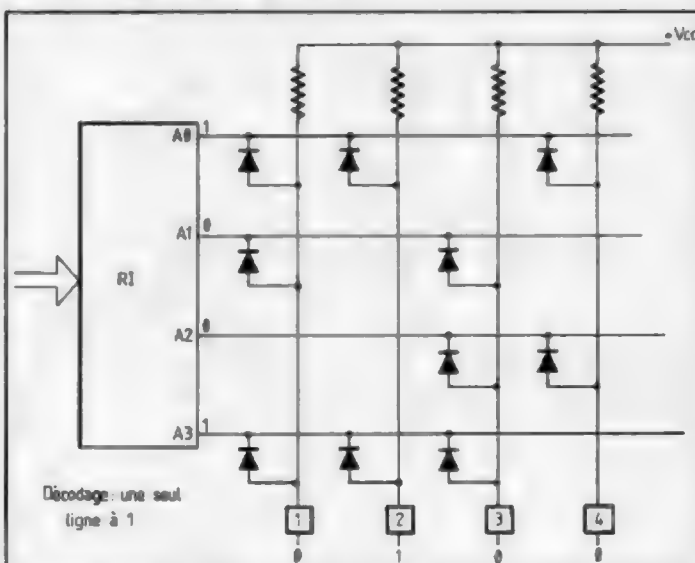


Fig. 29 : Décodeur à diodes : la matrice à diodes décode le mot 1001 placé dans le RI en mettant la ligne 2 à 1 et les autres à 0.

Malheureusement, la ligne 2 restera à 1 tant que A0 et A3 = 1, ce qui rend possible les cas où A1 = 0 et A2 = 0, soit les possibilités suivantes :

- 1001 ligne 2 = 1
- 1101 ligne 2 = 1
- 1111 ligne 2 = 1
- 1011 ligne 2 = 1

Pour éviter cette incertitude, on décode simultanément le code et son inverse.

Le décodage est dans ce cas parfaitement défini et correspond au code A3.A2.A1.A0 (fig. 30).

Une autre solution consiste à utiliser une matrice de transistors MOS disposés en fonction logique OU (fig. 31). Chaque MOS est commandé par la grille et constitue un point mémoire :

- si la couche d'oxyde est épaisse, le transistor est difficile à saturer
- si elle est mince, le transistor se sature facilement (fig. 32).

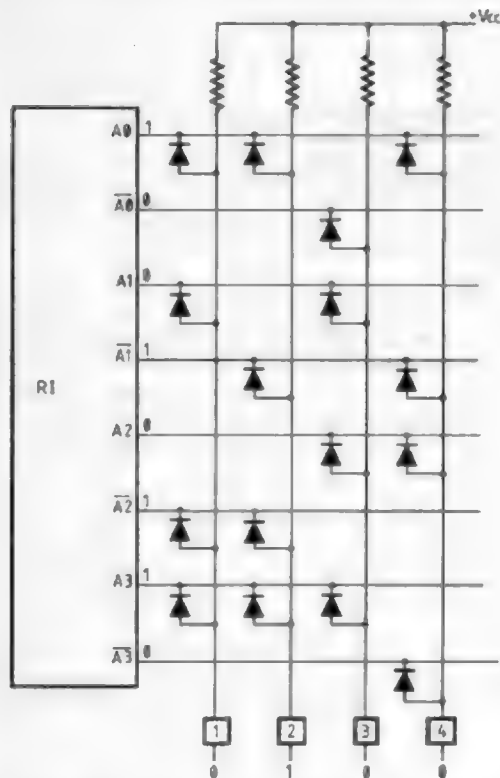


Fig. 30 : Décadeur à diodes : un seul cas possible pour un code donné.

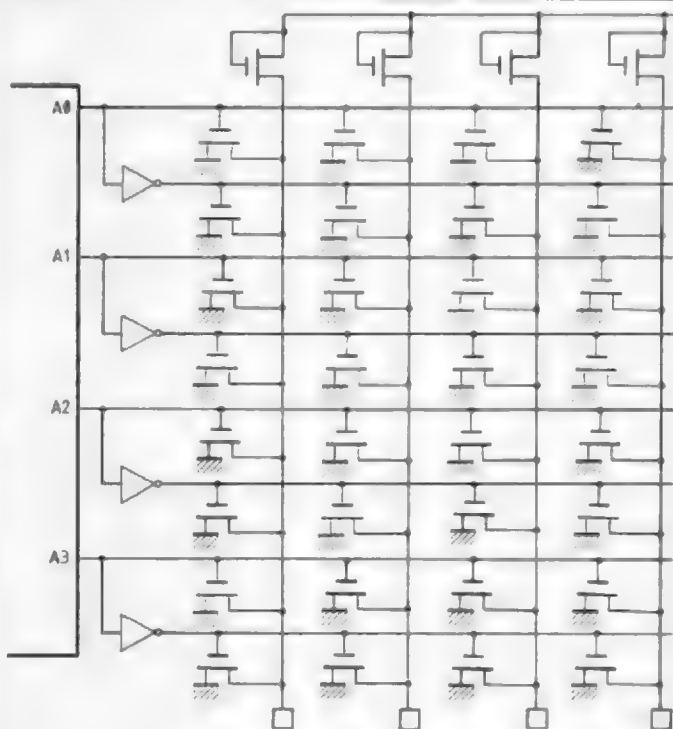


Fig. 31 : Décadeurs à transistors MOS.

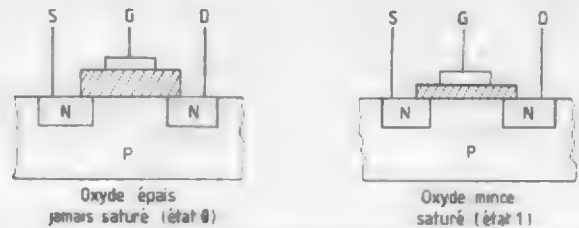


Fig. 32 : Point mémoire d'un décodeur.

Examinons le décodage de LDA (= \$86 = 10000110)

$$LDA = D7 \bar{D6} \bar{D5} \bar{D4} \bar{D3} D2 D1 \bar{D0}$$

La matrice de décodage étant constituée par des fonctions OU, nous pouvons transformer cette expression en utilisant le théorème de Morgan de façon à ne faire apparaître que des fonctions OU logiques :

$$LDA = D7 \cdot \bar{D6} \cdot \bar{D5} \cdot \bar{D4} \cdot \bar{D3} \cdot D2 \cdot D1 \cdot \bar{D0} = \bar{D7} + D6 + D5 + D4 + D3 + \bar{D2} + \bar{D1} + D0$$

$$\bar{LDA} = (\bar{D7} + \bar{D2} + \bar{D1}) + (D6 + D5 + D4 + D3 + D0)$$

Le schéma logique de  $\bar{LDA}$  est donné fig. 33 (les points mémoire 0 sont encadrés) :

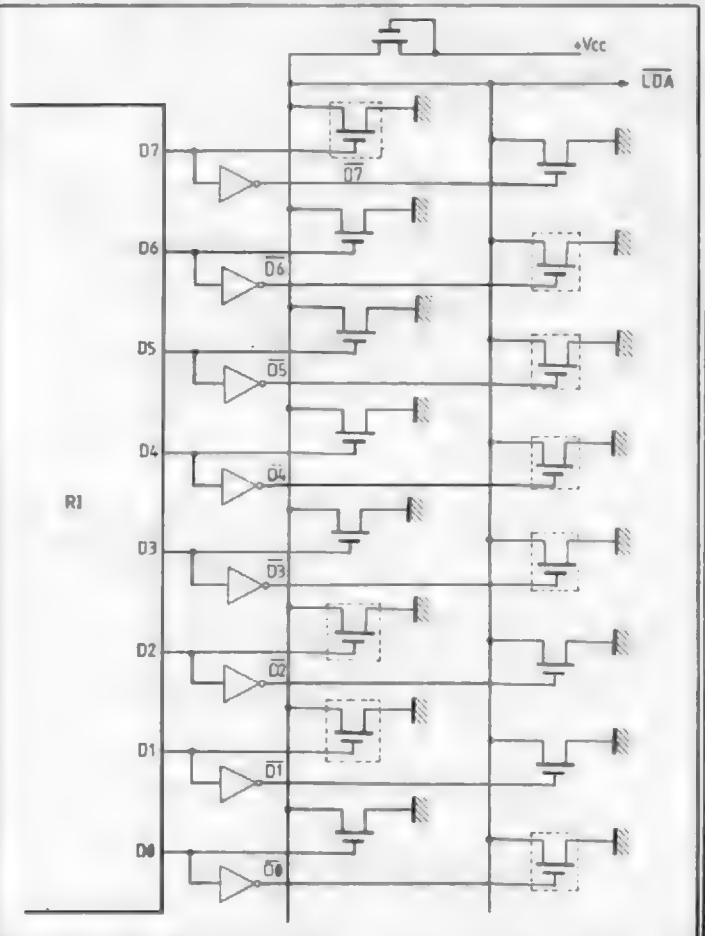


Fig. 33 : Fonction LDA sur une matrice MOS (un transistor encadré ne peut être saturé (état 0)).



## Décodeur en structure PLA (Programmable Logic Array)

Un réseau logique programmable est basé sur le principe suivant : toute fonction logique peut s'écrire sous forme d'une **somme de produits**.

Si on prend  $LDA = D7 \cdot \overline{D6} \cdot \overline{D5} \cdot \overline{D4} \cdot \overline{D3} \cdot D2 \cdot D1 \cdot \overline{D0}$ , on arrive à :

$$LDA = (D7 \cdot D2 \cdot D1) \cdot (\overline{D6} \cdot \overline{D5} \cdot \overline{D4} \cdot \overline{D3} \cdot \overline{D0})$$

$$LDA = (D7 \cdot D2 \cdot D1) \cdot (\overline{D6} \cdot \overline{D5} \cdot \overline{D4} \cdot \overline{D3} \cdot \overline{D0})$$

$$LDA = (D7 \cdot D2 \cdot D1) + (\overline{D6} \cdot \overline{D5} \cdot \overline{D4} \cdot \overline{D3} \cdot \overline{D0})$$

Nous avons une relation équivalente à une somme de produits. Il est alors possible de traduire cette équation par un schéma regroupant une matrice produit (ET logique) et une matrice somme (OU logique), fig. 34.

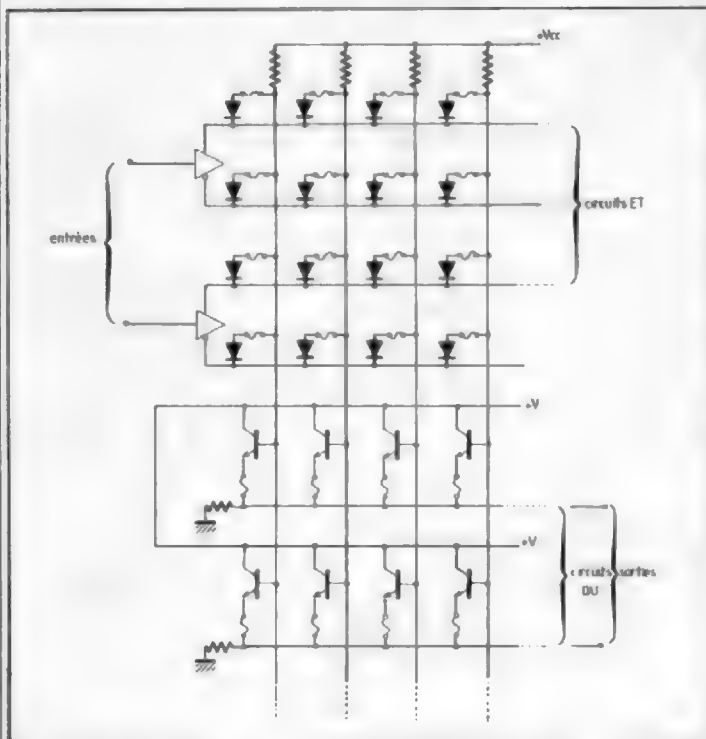


Fig. 34a : Une structure PLA à diodes et transistors à fusibles.

Les portes ET sont réalisées à l'aide d'un réseau à diodes programmables par l'utilisateur selon la technologie à fusibles.

A chaque entrée se trouve un inverseur pour avoir la variable et son complément, ce qui est très utile dans l'écriture des équations logiques.

Les portes OU sont réalisées par des transistors montés en émetteurs suiveurs.

Si on a par exemple un PLA à 8 entrées et 8 sorties, on réalisera  $2^8 \times 8 = 2^8 \times 2^3 = 2^{10} \times 2 = 2K$  combinaisons.

Ceci est donc équivalent à une mémoire ROM à 2K points mémoires ou de 2K bits.

Un PLA occupe donc moins de surface qu'une ROM à capacité égale et c'est cette solution qui sera choisie dans l'unité centrale du MOPET, mais au lieu d'utiliser des diodes, il sera très facile d'utiliser des MOS : dans ce cas, la matrice produit est constituée de NAND (fig. 35).

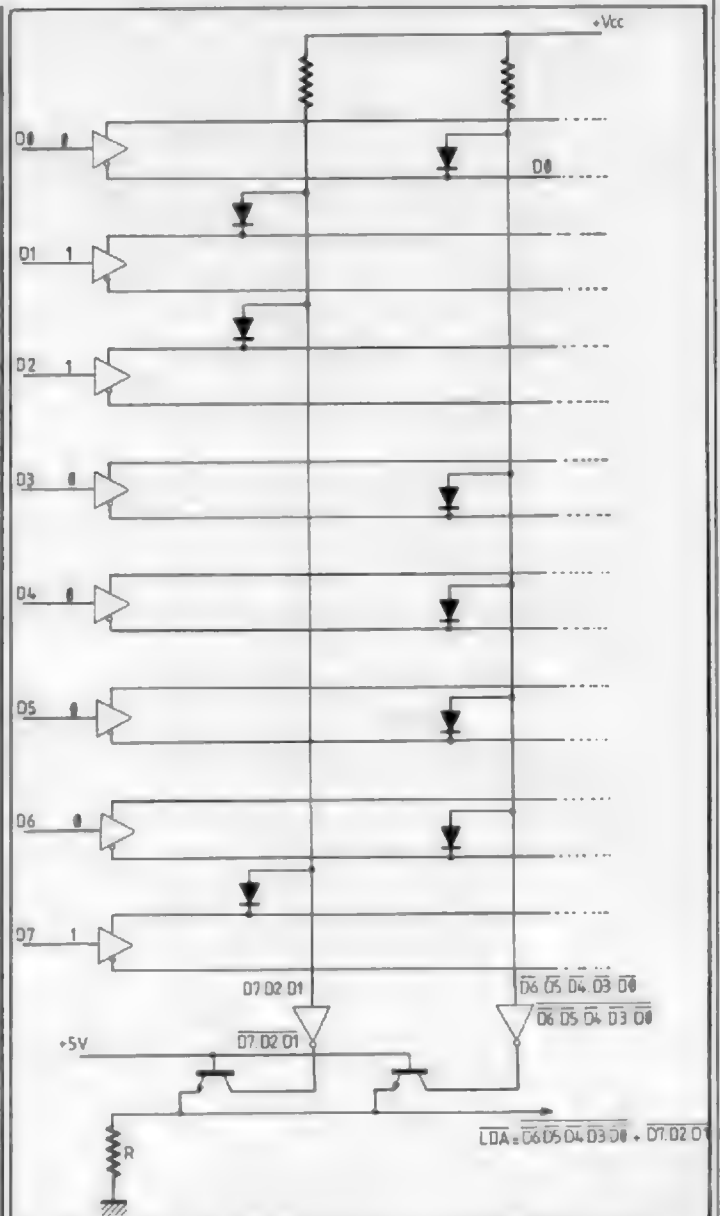


Fig. 34b : Fonction LDA dans une structure PLA à diodes fusibles.

## Le séquenceur

Il s'agit d'un élément très important de l'unité centrale. En effet, le séquenceur doit fournir :

- tous les signaux indispensables pour le milieu extérieur tels que :

- \* un Read/Write (Read = 1 Write = 0) pour lire une mémoire ou écrire.
- \* un signal périodique à la fréquence de l'horloge interne pour être synchrone avec l'extérieur (nous verrons plus loin que ce problème est un peu plus complexe).
- \* l'indication de l'état interne de l'unité centrale (fetch, execute, arrêt...)

Il doit aussi mettre l'unité centrale dans un certain état

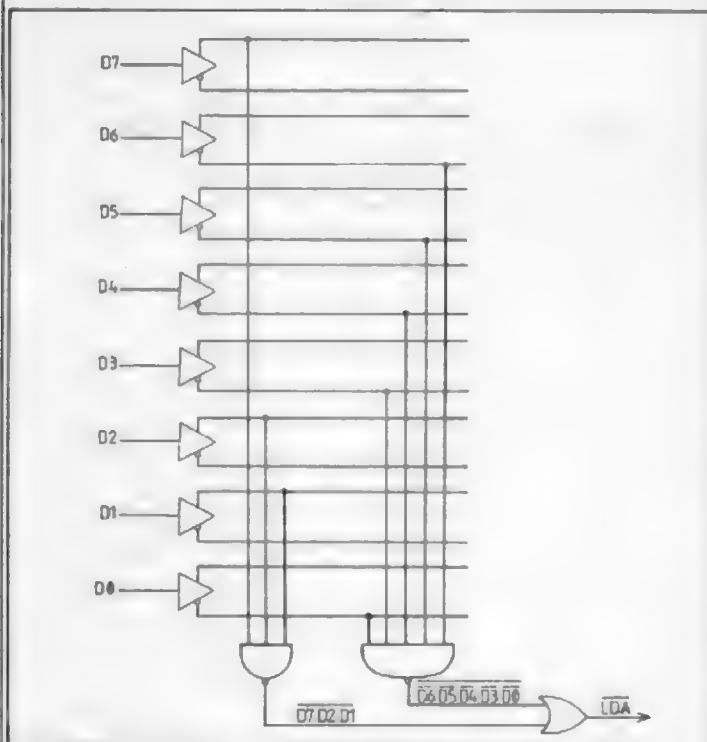


Fig. 35 : Représentation d'une structure PLA à NAND.

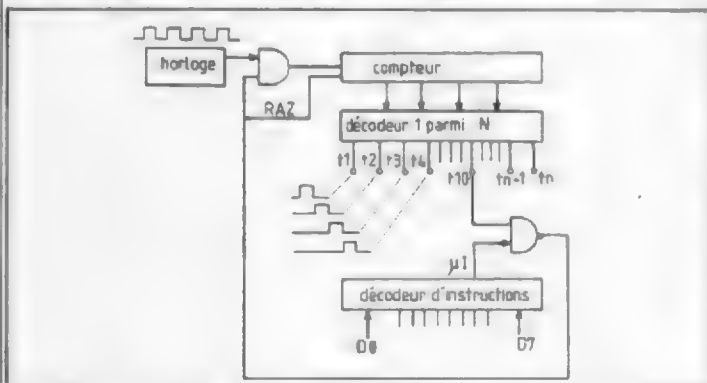


Fig. 36 : Principe d'un distributeur de phase. Si une instruction doit être exécutée en 10 périodes d'horloge, le distributeur sera bloqué à la 10<sup>e</sup> impulsion grâce à la porte NAND.

suivant la demande en provenance de l'extérieur :

- \* demande de mise en haute impédance de l'unité centrale
- \* demande de ralentissement de la fréquence d'horloge (pour des circuits lents)
- \* changement immédiat de processus pour exécuter en urgence un autre programme (interruptions).

Enfin, et c'est son rôle principal, il doit repérer les micro-instructions dans le temps.

Nous allons faire une analyse succincte de ce séquenceur :

Nous avons vu que l'exécution d'une instruction nécessite un certain nombre de micro-instructions élémentaires. Il est donc très important que chacune de ces micro-instructions, générées par le séquenceur, soit repérée dans le temps ; leurs phases doivent donc être émises par un distributeur de phase dont le principe est présenté fig. 36.

Des impulsions d'horloge sont appliquées à un compteur. Un décodeur 1 parmi N fournit successivement à chacune de ses sorties les phases t1, t2, t3... tn.

Lorsqu'une instruction est décodée, le décodeur d'instruction ■ une de ses lignes qui passe à 1 ( $\mu 1 = 1$ ).

Si on considère que cette instruction doit être exécutée en 10 coups d'horloge, il suffira de disposer une porte NAND avec t10 et  $\mu 1$  en entrée ; la sortie de la porte bloquant le compteur arrêtera automatiquement le distributeur de phase à la 10<sup>e</sup> impulsion d'horloge.

Il s'agit maintenant de mettre en équation toutes les micro-instructions de l'unité centrale.

Pour simplifier le processus, supposons que l'unité centrale est dans le cycle exécuté de : l'instruction LDA # \$3F (charge \$3F dans l'accumulateur A : l'instruction LDA est déjà décodée).

Au temps T1 : le PC s'incrmente d'un pas et pointe sur l'opérande ( $pc + 1 = 1$ ).

t2 : la ligne Read passe à 1 pour lire le contenu de la mémoire = 3F,

t3 : le contenu de pc est placé sur le bus d'adresses  $pco = 1$ , Read = 1, on lit donc le contenu

pointé par pc,

t4 : l'accumulateur A est chargé ( $Accin = 1$ ,  $pco = 1$ , Read = 1),

t5 : le pc s'incrmente d'un pas (instruction suivante), la ligne Read est désactivée.

L'opérande LDA # \$3F est terminée, ce qui nous mène aux équations suivantes :

$$pc + 1 = LDA.(t1 + t5)$$

$$pco = LDA.(t3 + t4)$$

$$Read = LDA.(t2 + t3 + t4)$$

$$Accin = LDA.t4$$

et au schéma de la figure 37.

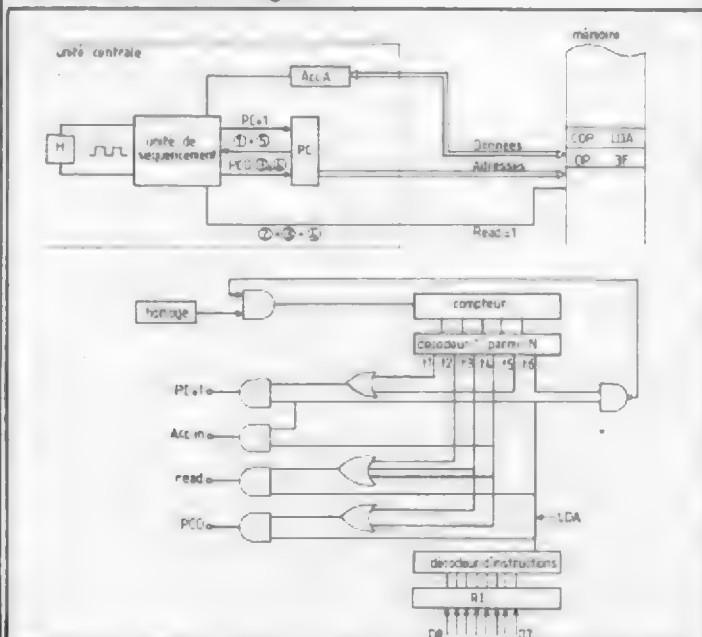


Fig. 37 : Exemple des micro-instructions à générer pour l'instruction LDA (chargement d'une valeur dans l'accumulateur).

Bien sûr, il faudra étudier chacune des micro-instructions pour l'ensemble des instructions.

Par exemple, si nous regardons l'équation de pco pour LDA # \$ 12, on trouvera une équivalence avec ADDA # \$ 20, CMPA # \$ 12, etc.

pco = LDA (t3 + t4)  
pco = ADDA (t3 + t4)  
pco = CMPA (t3 + t4)

pco peut alors s'écrire  $pco = (LDA + ADDA + CMPA) \cdot (t3 + t4)$ .

On arrive ainsi à un ensemble d'équations représentatives de toutes les micro-instructions de l'unité centrale.

En remplaçant tout cela par un PLA on arrivera à simplifier d'une façon notable le schéma d'un séquenceur.

## Le cheminement des informations dans une unité centrale

Nous allons expliquer le cheminement des informations dans une unité centrale.

Il ne s'agira pas de la réalité mais d'une approche car le constructeur ne fournit pas d'explications détaillées sur ce sujet et il faut bien le comprendre !

L'unité centrale doit fournir à l'extérieur au minimum :

- un bus de données bi-directionnel
- un bus d'adresses uni-directionnel
- une ligne de lecture/écriture
- une horloge mettant les périphériques en phase avec le travail de l'unité centrale.

Notons que la ligne READ/WRTIE, les bus de données et d'adresses doivent pouvoir se mettre en haute impédance. Afin de rendre les micro-instructions totalement transparentes pour le programmeur, on utilisera deux horloges distinctes : une horloge interne ( $\phi_{int}$ ) synchronisant les micro-instructions et une horloge externe ( $f$ ) dont une période représentera le travail de toute une séquence.

Si l'horloge interne a une fréquence quatre fois plus élevée que l'horloge externe, on aura huit périodes d'horloge interne soit huit micro-instructions à générer.

Afin de bien comprendre, nous allons exécuter le programme suivant :

COP COP

```
$1000 LDA # $3F      86  3F
$1002 STA  $5000     B6 50 00
$005
```

Soit à charger l'accumulateur A avec la valeur 3F (86 = COP 3F = OP) puis à stocker le contenu de A à l'adresse \$5000 (B6 = COP 5000 = OP). Le programme se trouve localisé à l'adresse \$1000.

La figure 38 montre les différentes micro-instructions à générer au sein de l'unité centrale pour ces deux lignes de programme.

Remarquons que l'horloge de 4 MHz fabrique deux signaux :  $\phi_{int} = 125$  ns et une période de  $E = 500$  ns. Les buffers d'adresse, de données et la ligne R/W sont à trois états (signal TSC = Three State Control), le signal latch sert à bloquer les lignes dans un état indéterminé.

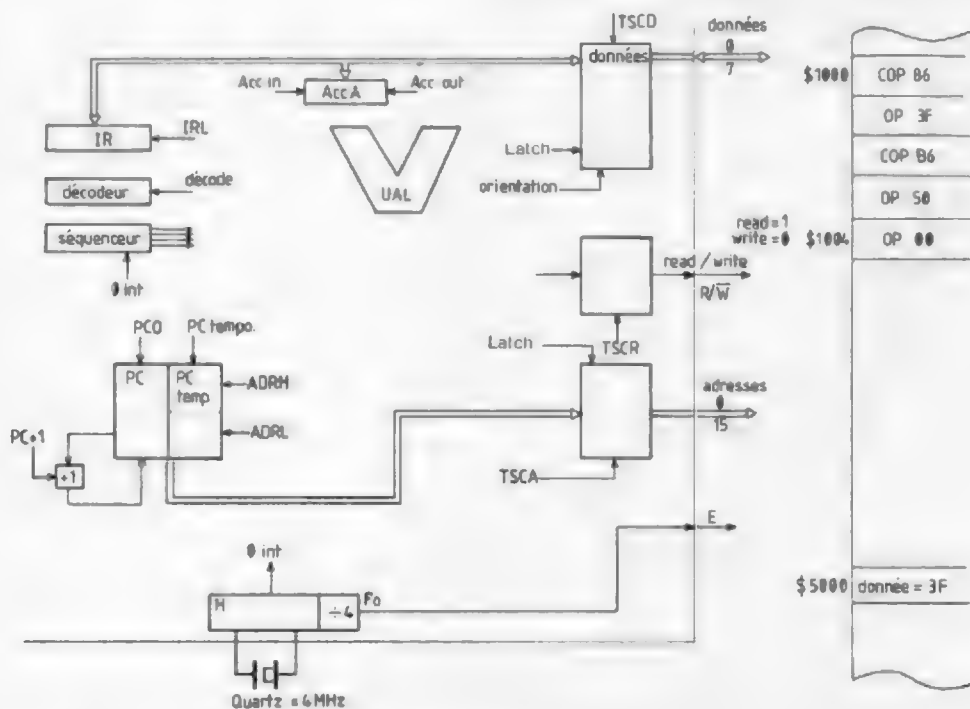


Fig. 38 : Micro-instructions à générer par l'unité centrale pour LDA # \$3F et STA \$5000.



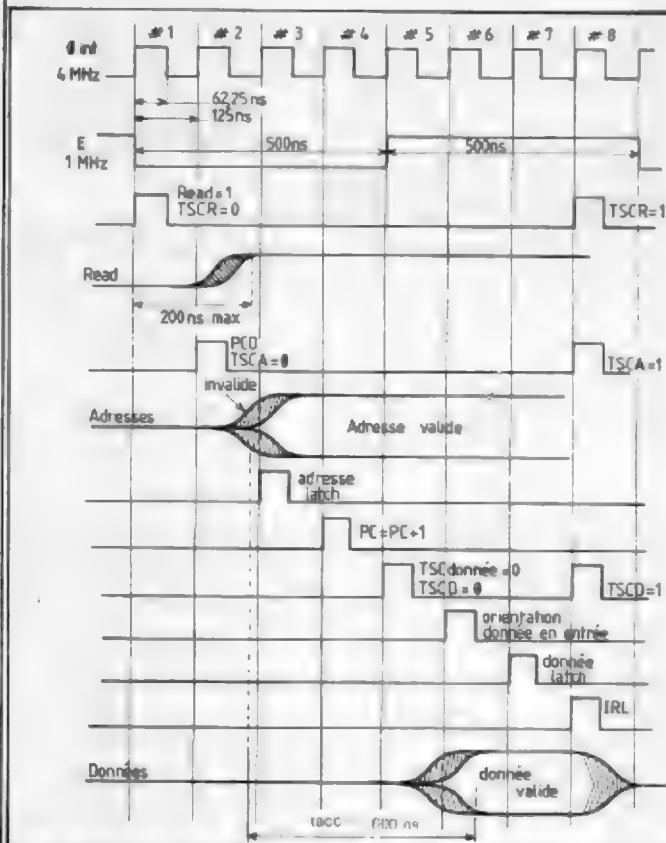


Fig. 39 : 1<sup>re</sup> séquence de LDA # \$3F.

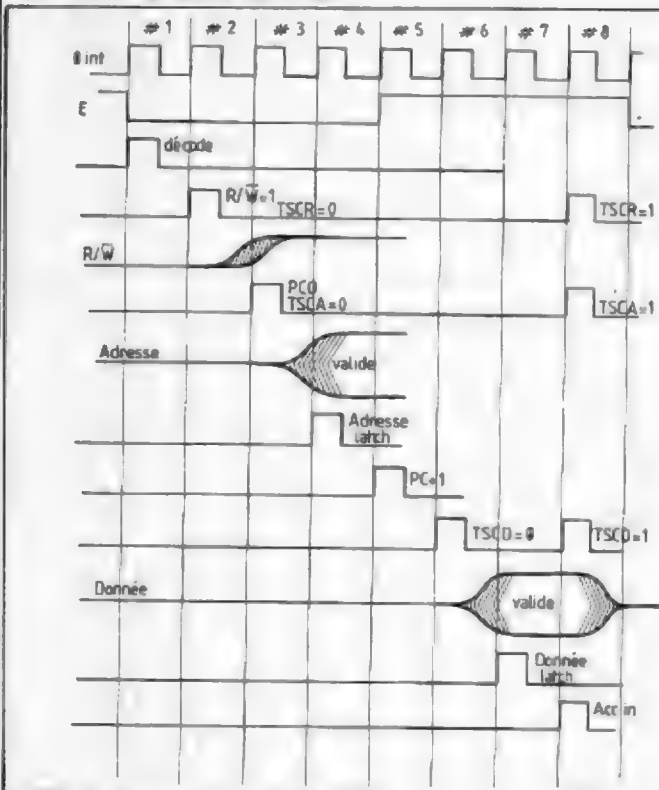


Fig. 40 : 2<sup>e</sup> séquence de LDA # \$3F.

### 1<sup>re</sup> séquence de LDA # \$3F (fig. 39) :

On supposera que le compteur de programme PC contient au départ la valeur \$ 1000.

**Période n°1 de  $\phi$  int :** on ouvre la ligne R/W (TSCR = 0) et on la positionne à 1, la sortie R/W passe donc à 1 après un certain délai dû à l'établissement de la tension.

**Période n°2 de  $\phi$  int :** on ouvre le compteur de programme (PC output = PCO) ainsi que le buffer d'adresse (TSCA = 0).

Après un certain délai, les adresses sont valides (stables), sur le bus d'adresse.

**Période n°3 de  $\phi$  int :** une impulsion latch sur le buffer d'adresse maintient le bus dans un état stable.

**Période n°4 de  $\phi$  int :** ce qui permet d'incrémenter le compteur de programme (impulsion PC + 1) afin de pointer ultérieurement à l'adresse \$1001.

**Période n°5 de  $\phi$  int :** E passe à 1 et y restera jusqu'à la fin de la 8<sup>e</sup> période de  $\phi$  int.

Durant cette période, on ouvre le buffer de données TSCD = 0.

**Période n°6 de  $\phi$  int :** on oriente le buffer de données en entrées (puisque on fait une lecture de la mémoire).

**Période n°7 de  $\phi$  int :** la donnée doit être présente sur le bus de données, il suffit donc de la «latcher» dans le buffer.

**Période n°8 de  $\phi$  int :** la donnée étant présente dans le buffer, il suffit de la charger dans le registre d'instruction IR (impulsion Instruction Register Load = IRL) puis de mettre tous les buffers en haute impédance pour finir la séquence TSCR = 1 TSCA = 1 TSCD = 1.

### 2<sup>e</sup> séquence de LDA # \$3F (fig. 40) :

**Période n°1 de  $\phi$  int :** une impulsion «décode» permet de décoder le code opératoire reçu (86) se trouvant dans le registre d'instruction IR. Il s'agit de charger l'accumulateur A avec la valeur pointée par le compteur de programme PC.

**Période n°2 de  $\phi$  int :** on ouvre la ligne R/W et on la positionne à 1.

**Période n°3 de  $\phi$  int :** on valide le compteur de programme et on ouvre son buffer.

**Période n°4 de  $\phi$  int :** la valeur de PC (\$1001) se trouve «latchée» dans le buffer ; à ce moment, l'adresse est validée sur son bus.

**Période n°5 de  $\phi$  int :** le compteur de programme s'incrémente d'un pas et passe donc à \$1002.

**Période n°7 de  $\phi$  int :** c'est durant ce temps que l'on considère qu'une donnée est validée sur le bus, il suffit donc de la «latcher» dans le buffer. En l'occurrence, il s'agit ici de l'opérande \$3F puisque l'on pointe sur l'adresse \$1001.

**Période n°8 de  $\phi$  int :** une impulsion Accin charge l'accumulateur A avec la donnée se trouvant dans le buffer. Les bus sont à nouveau fermés.

**7<sup>e</sup> phase :** la donnée est «latchée». On écrit donc la valeur \$3F à l'adresse \$5000.

**8<sup>e</sup> phase :** on ferme les buffers et puisque PC = \$1005, on pourra exécuter une nouvelle instruction à la séquence suivante.

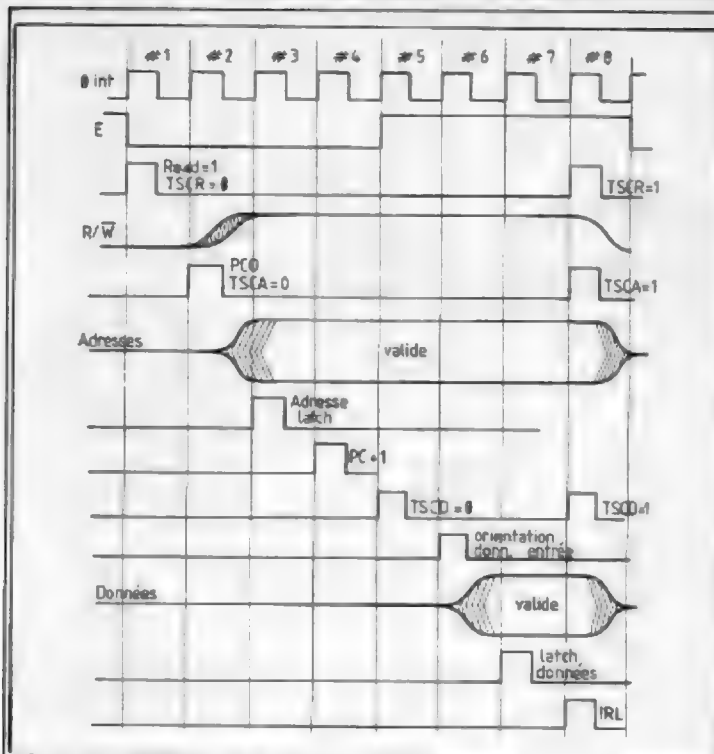


Fig. 41 : 1<sup>re</sup> séquence de STA \$5000.

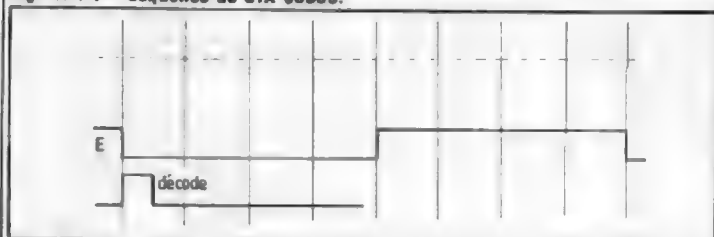


Fig. 42 : 2<sup>e</sup> séquence de STA \$5000.

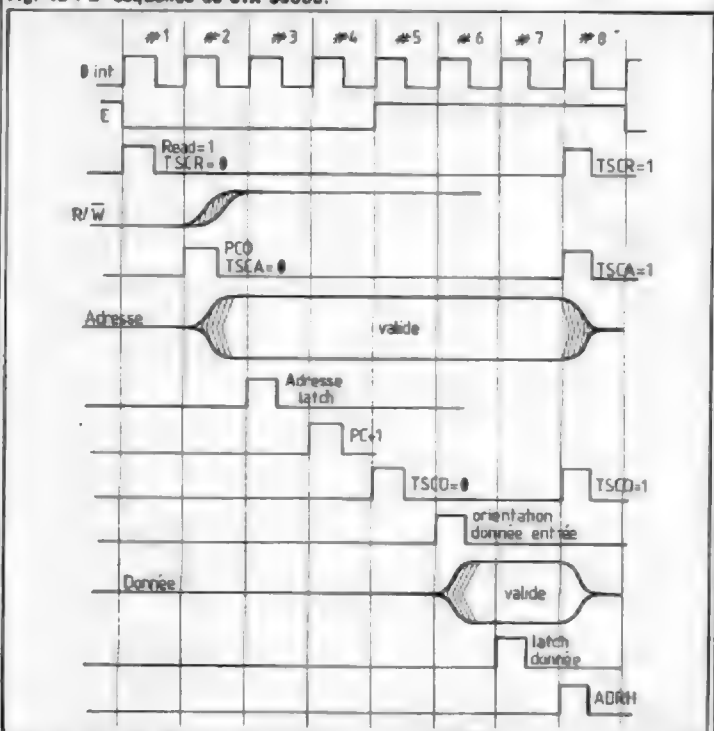


Fig. 43 : 3<sup>e</sup> séquence de STA \$5000.

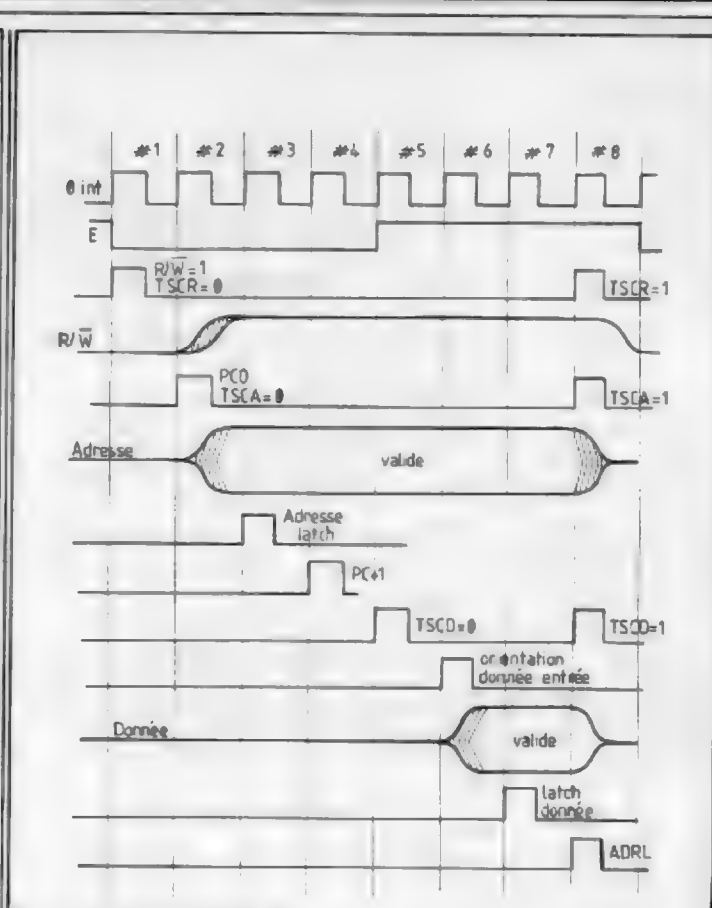


Fig. 44 : 4<sup>e</sup> séquence de STA \$5000.

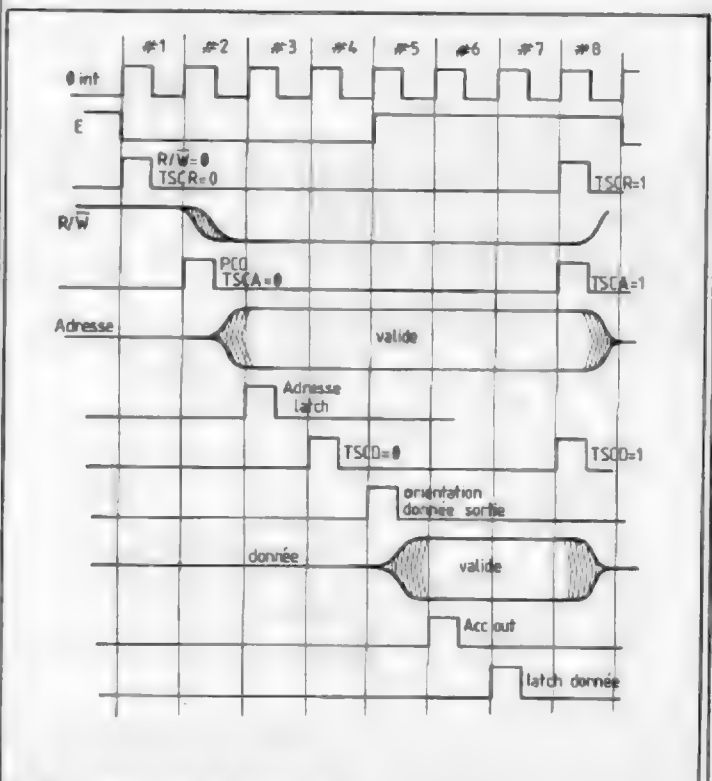


Fig. 45 : 5<sup>e</sup> séquence de STA \$5000.

## Conclusion

L'instruction STA \$5000 s'est exécutée en cinq périodes d'horloge E soit 5  $\mu$ s.

On remarque à nouveau qu'une adresse est toujours valide lorsque E = 1 et qu'une donnée est toujours disponible à la fin de E = 1.

De plus, l'unité centrale passe finalement la majorité de son temps à lire.

En fait, on remarque sur les figures 39 à 45 qu'une adresse est stable à partir de la 3<sup>e</sup> phase de  $\emptyset$  int.

Si l'on fabrique une nouvelle horloge Q en quadrature sur E et de même fréquence, on pourrait poser pour postulat :

- qu'une adresse est valide sur le front montant de Q
- la ligne R/W est également valide sur le front montant de Q

On aboutit ainsi aux schémas des figures 46 et 47 qui représentent l'état des bus d'adresses et de données pour une lecture et une écriture.

**Remarque :** dans les exemples que nous avons cités, le cheminement des micro-instructions en fonction des phases de  $\emptyset$  int n'est qu'une vue de l'esprit en comparaison avec la réalité mais cela donne une idée du fonctionnement interne de toute unité centrale.

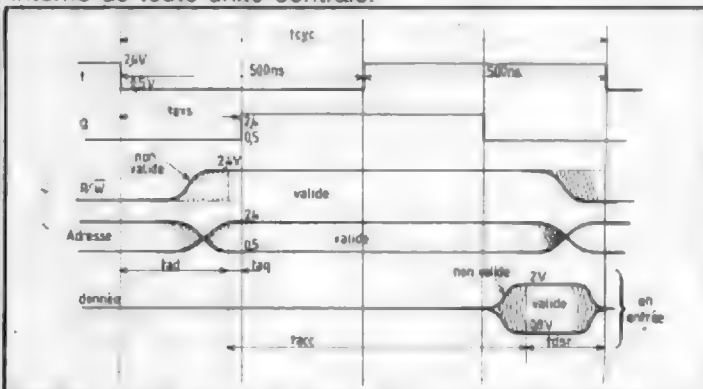


Fig. 46 : Lecture de données en mémoire ou en provenance de périphériques.  
 $t_{cyc}$  : temps de cycle = durée d'une séquence (si  $t = 1 \text{ MHz}$  -  $t_{cyc} = 1 \mu$ s.)  
 $t_{ave}$  : temps entre 1 Bas et Q haut : 250 ns max  
 $t_{ad}$  : temps de retard des adresses : 200 ns max  
 $t_{acc}$  : temps d'accès à la lecture : temps que mettra l'unité centrale pour acquérir une donnée : 895 ns min.  
 $t_{d\acute{e}}$  : temps entre adresse et Q haut : 25 ns min.  
 $t_{ds}$  : temps d'établissement des données en lecture : 80 ns min.

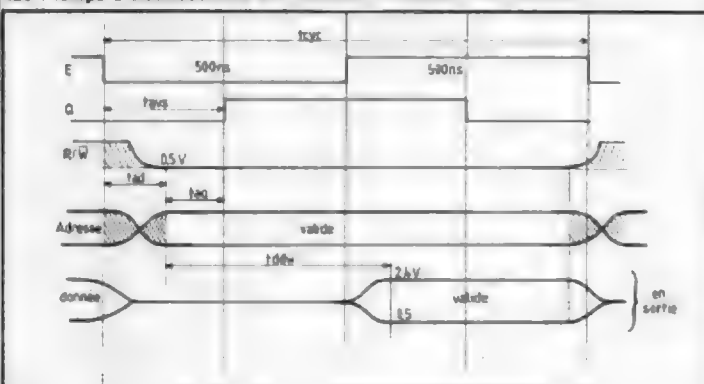


Fig. 47 : Écriture de données en mémoire ou dans les périphériques.  
 $t_{adw}$  : temps de retard des données : temps que mettra l'unité centrale pour envoyer une donnée : 225 ns max.

## Sélection mémoire

Compte tenu des remarques précédentes, nous remarquons qu'il est possible de sélectionner une mémoire durant la moitié d'un cycle d'horloge de l'unité centrale lorsque E = 1 puisque c'est durant ce moment-là que l'on dispose des adresses et des données (à condition que la ligne R/W soit à un état significatif).

Si l'on dispose d'une mémoire dont le temps d'accès est inférieur à 500 ns, il est donc possible de sélectionner celle-ci comme le montre la figure 48. Un décodeur/démultiplexeur sélectionne une mémoire (une RAM par exemple), dans une plage d'adresses qui correspond à sa capacité et ceci, durant le temps où E = 1.

Puisque le bus d'adresse est stable durant cette demi-période, la mémoire acceptera ou fournira la donnée au bout d'un temps minimal  $t_{acc}$ .

La ligne R/W permettra d'orienter le bus de données en entrée ou en sortie.

### Conclusion :

Une instruction LDA #\$3F est donc exécutée en deux séquences (deux périodes de E) soit, dans notre exemple, en 2  $\mu$ s.

On remarque également que c'est pendant E = 1, qu'une adresse est valide sur le bus d'adresse et c'est à la fin de E = 1 qu'une donnée est présente sur son bus.

Continuons nos investigations en passant à la deuxième

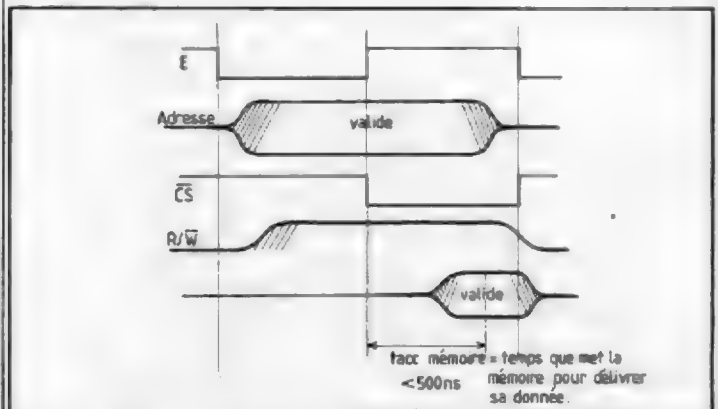
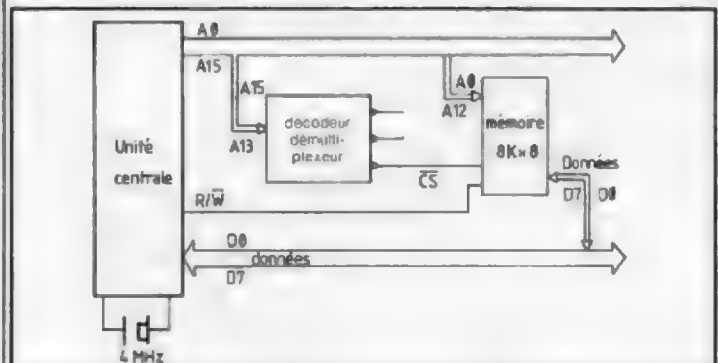


Fig. 48 : Sélection d'une mémoire RAM en lecture.



ligné de programme.

Il s'agit de stocker le contenu de l'accumulateur A à l'adresse \$5000.

#### 1<sup>ère</sup> séquence de STA \$5000 (fig. 41) :

Elle ressemble comme une soeur à la première séquence de LDA#\$3F, ce qui est normal puisqu'il s'agit d'acquérir le code opératoire (B6) afin de le décoder.

#### 2<sup>ème</sup> séquence de STA \$5000 (fig. 42) :

Il s'agit d'une séquence presque perdue puisqu'elle ne sert qu'à décoder l'instruction.

#### 3<sup>ème</sup> séquence de STA \$5000 (fig. 43) :

Cette séquence est à nouveau classique puisqu'elle ressemble à la première : la donnée reçue (\$50 puisque le compteur de programme pointe à l'adresse \$1003) est chargée dans la partie haute du registre temporaire d'adresse (ADRH).

Remarquons qu'il ne faut surtout pas charger le compteur de programme avec la valeur \$5000 car on perdra définitivement la valeur correspondant à la suite du programme !

#### 4<sup>ème</sup> séquence de STA \$5000 (fig. 44) :

Cette séquence ressemble à la troisième. On charge la donnée (\$00 puisque PC pointe en \$1004) dans la partie basse du registre temporaire d'adresse (impulsion ADRL). A l'issue de cette séquence, le registre temporaire d'adresses contient la valeur \$5000.

#### 5<sup>ème</sup> séquence de STA \$5000 (fig. 45) :

**1<sup>ère</sup> phase** : on positionne la ligne R/W à 0 afin d'opérer une écriture en mémoire.

**2<sup>ème</sup> et 3<sup>ème</sup> phase** : on délivre le contenu du registre temporaire sur le bus d'adresses.

**4<sup>ème</sup> et 5<sup>ème</sup> phase** : le buffer de données est orienté dans le sens Unité Centrale-Mémoire.

**6<sup>ème</sup> phase** : le contenu de l'accumulateur est délivré sur le bus de données (impulsions Accout).

## Le 6809

L'unité centrale que nous allons étudier durant les chapitres qui suivent porte la référence EF 6809, elle est fabriquée par la société Thomson-EFCIS, le créateur étant Motorola. Ce microprocesseur existe en trois versions :

- EF 6809 fonctionne avec une fréquence d'entrée de 4 MHz, et délivre donc une fréquence E = 1 MHz ;
- EF 68 A 09 avec f = 6 MHz et E = 1,5 MHz ;
- EF 68 B 09 avec f = 8 MHz et E = 2 MHz.

La puissance dissipée est de 1 W max. (sous 5 V).

La figure 49b nous donne le brochage de ce composant, on note qu'il comprend :

Le 6809 comprend cinq registres internes de 16 bits et 4 registres de 8 bits (fig. 49a), ils seront décrits en détail dans le chapitre suivant.

**Nota** : la suite de ce chapitre est tirée de documents EFCIS.

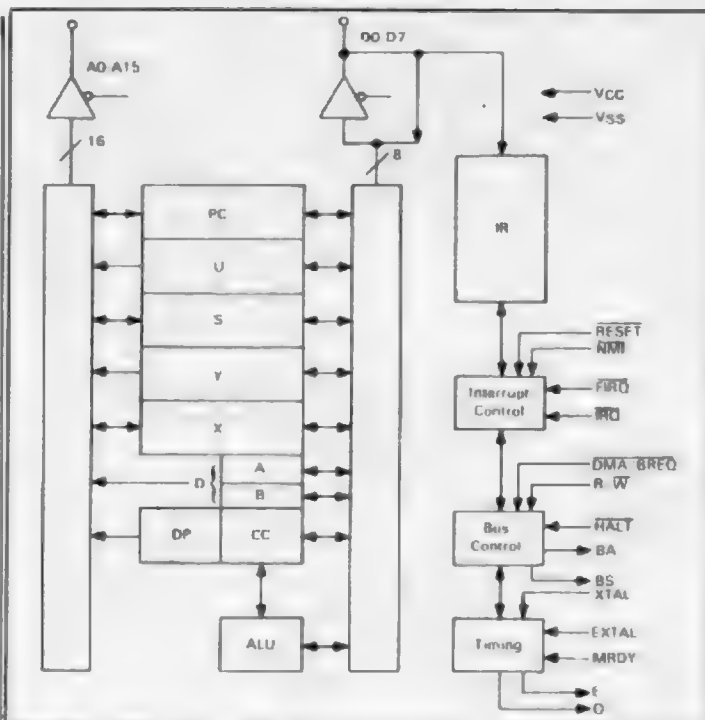


Fig. 49a : Synoptique du 6809.

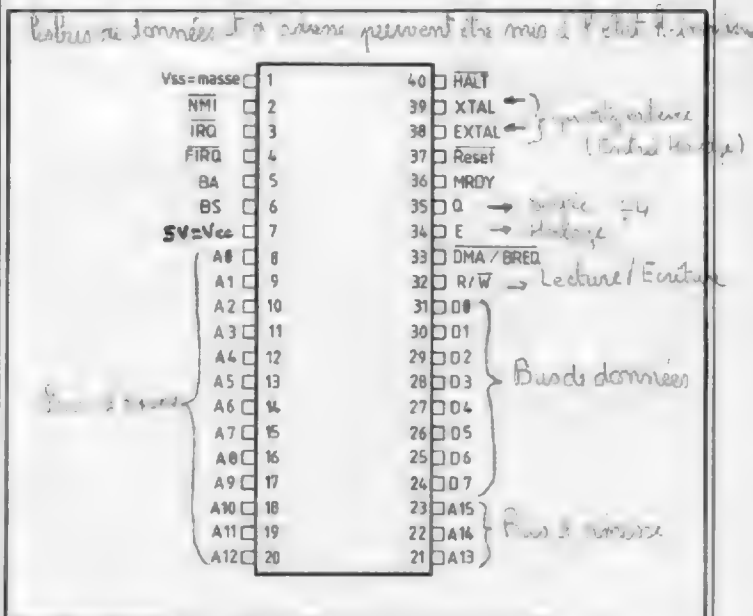


Fig. 49b : Brochage du 6809.

- un bus d'adresses (A0-A15) : lorsque le bus n'est pas utilisé par le 6809 pour un transfert de données, il sort l'adresse \$FFFF R/W = 1 et BS = 0.

Nous avons vu que les adresses sont validées sur le front montant de Q. Tous les amplificateurs du bus d'adresses sont mis à l'état haute impédance lorsque la sortie BA est à l'état haut.

- un bus de données (D0-D7) : il est bidirectionnel et sert à la transmission de données 8 bits ;
- une ligne de lecture-écriture (R/W) : cette ligne indique le sens du transfert des données sur le bus données. Un niveau bas indique que l'unité centrale procède à une écriture, R/W passe à l'état haute impédance lorsque BA est à l'état haut. R/W est validé sur le front montant de Q.
- une ligne d'initialisation (RESET) : un niveau bas sur cette entrée trigger de Schmitt durant un temps supérieur à un cycle bus provoque une initialisation de l'unité centrale (fig. 50). Les vecteurs d'initialisation seront accessibles aux adresses \$FFFE et \$FFFF dès lors que la condition logique BA = 0 et BS = 1.

A la mise sous tension, cette ligne doit être maintenue à l'état bas jusqu'à ce que l'oscillateur d'horloge ait atteint un régime de fonctionnement normal.

Un simple réseau RC peut être utilisé pour initialiser l'ensemble du système puisque l'entrée Reset du EF 6809 possède un trigger de Schmitt ayant une tension de seuil supérieure à celle des périphériques standards. Ce seuil de tension plus élevé garantit que tous les périphériques ne sont pas en phase d'initialisation après le processeur.

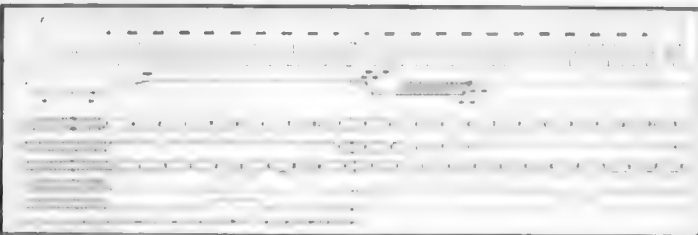


Fig. 50 : Fonctionnement d'un RESET.

- une ligne d'arrêt (HALT) : un niveau bas sur cette entrée provoque l'arrêt du microprocesseur à la fin de l'instruction en cours et celui-ci demeure à l'arrêt indéfiniment sans perte de données. A l'arrêt, la sortie BA passe à l'état haut indiquant que les bus sont à l'état haute impédance. BS est aussi à l'état haut indiquant que le processeur est arrêté ou à l'état bus accordé. A l'état arrêt, le microprocesseur ne répond pas à des demandes externes en temps réel (FIRQ, IRQ) bien que DMA/BREQ soit toujours accepté, et que NMI et RESET soient mémorisées pour une réponse ultérieure. A l'état arrêt, Q et E continuent à fonctionner normalement. Si le microprocesseur est arrêté (RESET, DMA/BREQ), l'état HALT (BA et BS = 1) peut être atteint lorsque l'entrée Halt est mise à l'état bas bien que l'entrée Reset soit encore à l'état bas. Si DMA/BREQ et HALT sont tous les deux à l'état bas, le processeur continuera jusqu'au dernier cycle de l'instruction sur lequel le processeur sera arrêté (fig. 51).

- deux lignes d'état du bus (BA = Bus Available, BS = Bus Status : BA = Bus accordé ou Bus libre, BS = état du bus.

La sortie BA indique qu'un signal de commande interne fait passer les bus du microprocesseur à l'état haute impédance. Ce signal n'implique pas que le bus soit disponible pendant plus d'un cycle.

Lorsque BA passe à l'état bas, un cycle perdu supplémen-



Fig. 51 : Fonctionnement du halt et exécution d'une seule instruction.

taire se déroule avant que le microprocesseur n'occupe le bus.

Le signal de sortie état du bus, lorsqu'il est décodé avec BA, représente l'état du microprocesseur (validé sur le front montant de Q) :

BA	BS	Etat du microprocesseur
0	0	Fonctionnement normal
0	1	Reconnaissance d'interruptions
1	0	Reconnaissance de l'instruction SYNC
1	1	Arrêt ou Bus accordé

Une reconnaissance d'interruption (BA = 0, BS = 1) est présentée durant les deux cycles d'acquisition du vecteur d'interruption (RESET, NMI, IRQ, SWI, SW12, SW13). Cet état détecté et le décodage des quatre lignes d'adresses de poids faibles indiquent à l'utilisateur quel est le niveau d'interruption pris en compte et permet une vectorisation par les périphériques.

Il y a reconnaissance de synchronisation lorsque le microprocesseur rencontre l'instruction de synchronisation (SYNC), celle-ci est indiquée par BA = 1 et BS = 0, signifiant que le microprocesseur est en attente de synchronisation extérieure par l'intermédiaire d'une ligne d'interruption. La condition BA = BS = 1 est vraie lorsque le microprocesseur est dans l'état Halt ou bus accordé (ligne Halt = 0).

- une ligne d'interruption Non Masquable (NMI = Non Masquable Interrupt) : un front descendant sur cette entrée entraîne une séquence d'interruption non masquable. Une interruption non masquable ne peut pas être inhibée par programme et possède une priorité supérieure à FIRQ, IRQ ou aux interruptions logicielles. Lors d'une reconnaissance de NMI, l'état complet du microprocesseur est sauvegardé sur la pile. Après initialisation, une NMI ne sera prise en compte qu'après le premier chargement par programme du pointeur de pile S.

La largeur d'impulsion de NMI, à l'état bas, doit être au moins d'un cycle E. Si l'entrée NMI n'a pas un temps d'établissement suffisant en regard de Q, l'interruption ne sera prise en compte qu'au cycle suivant (fig. 52).

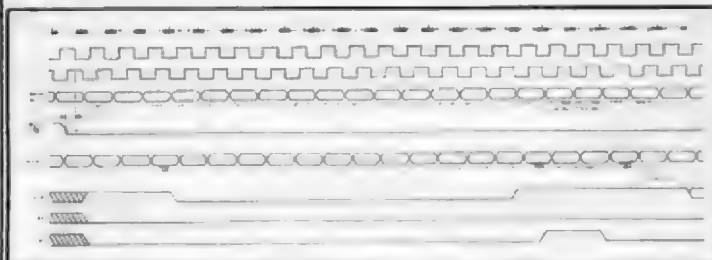


Fig. 52 : Exécution de TRQ ou RTI.

– une ligne de demande d'interruption rapide ( $\overline{\text{FIRQ}}$  = Fast Interrupt Request) : un niveau bas sur cette broche entraîne la séquence d'interruption rapide, à condition que le bit masque F du registre de condition soit à 0. Cette interruption a priorité par rapport à une demande d'interruption standard IRQ, elle est plus rapide puisqu'il n'y a sauvegarde sur la pile que du registre code condition et du compteur de programme. Le sous-programme de traitement des interruptions doit libérer la source d'interruption avant l'exécution de l'instruction RTI (fig. 53).

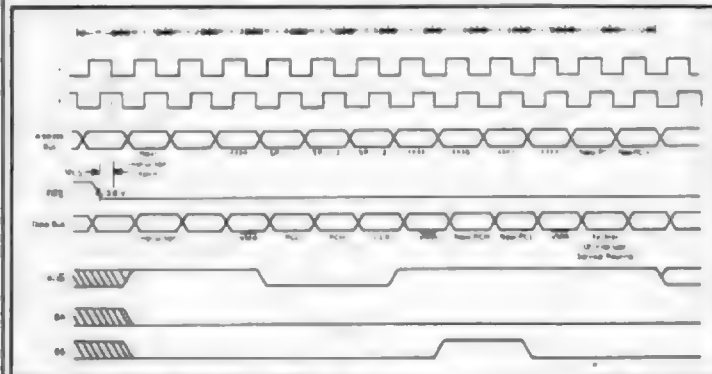
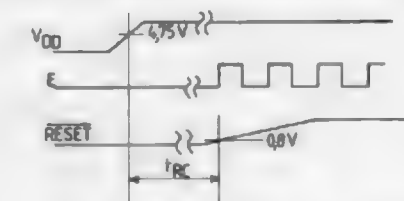


Fig. 53 : Exécution de l'interruption FIRQ.

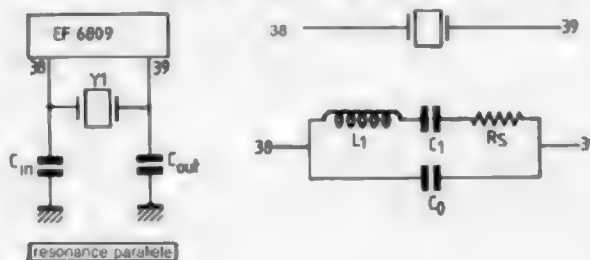
– une ligne de demande d'interruption (IRQ = Interrupt Request) : un niveau bas appliqué à cette entrée entraîne la séquence de traitement d'interruption  $\overline{\text{IRQ}}$ , à condition que le bit masque I du registre de condition soit à 0. Cette séquence réalisant la sauvegarde de l'état complet du processeur, la réponse sera plus lente que pour le FIRQ. Par ailleurs, IRQ a une priorité plus basse que FIRQ. Là encore, le sous-programme de traitement des interruptions doit libérer la source d'interruption avant d'exécuter l'instruction RTI (fig. 52).

– deux lignes XTAL et EXTAL : ces broches d'entrée sont utilisées pour connecter l'oscillateur interne à un quartz externe à résonance parallèle. Par ailleurs, la broche EXTAL peut être utilisée comme une entrée niveau TTL pour une horloge extérieure en mettant XTAL à la masse. Le quartz ou la fréquence externe est quatre fois la fréquence bus (fig. 54). Les règles d'implantation propres aux circuits RF doivent être observées dans le tracé des circuits imprimés.

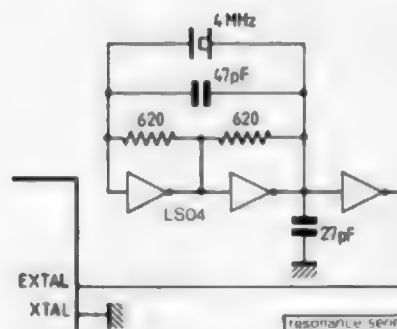
– deux lignes E et Q : E est identique au signal d'horloge  $\phi_2$  du EF 6800 ; Q est un signal d'horloge en quadrature qui pilote E.



	4 MHz	6 MHz	8 MHz
RS	50 $\Omega$	30-50 $\Omega$	20-40 $\Omega$
C0	6,5 pF	4,6 pF	4,6 pF
C1	0,025 pF	0,01-0,02 pF	0,01-0,02 pF
Q	> 30 K	> 20 K	> 20 K



résonance parallèle



résonance série

Fig. 54 : Connexion du quartz et démarrage de l'oscillateur.

Q n'a pas d'équivalent sur le EF 6800. Les adresses du microprocesseur sont validées sur le front montant de Q. Les données sont mémorisées sur le front descendant de E.

Le diagramme des temps pour E et Q est montré figure 55.

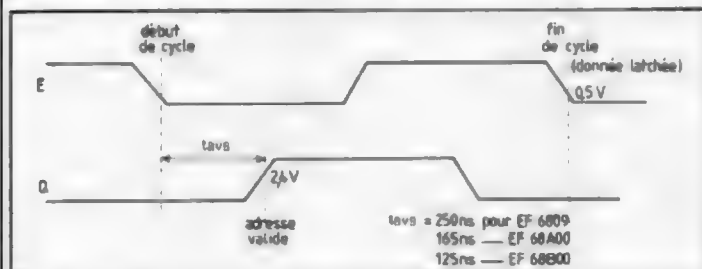


Fig. 55 : Relation entre E et Q.

– une ligne Memory Ready (MRDY = mémoire prête) : cette entrée de commande permet l'allongement de E pour augmenter le temps d'accès aux données. Lorsque MRDY est à l'état haut, E est en fonctionnement normal. Lorsque MRDY est à l'état bas, E peut être allongé de multiples



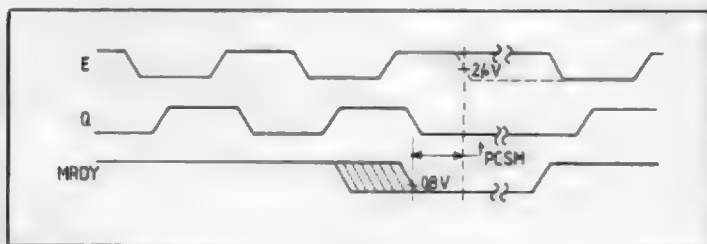


Fig. 56a : MRDY Timing.

entiers de 1/4 de cycle bus, permettant ainsi l'utilisation de mémoires lentes comme indiqué figure 56.

L'allongement maximum est de 10 microsecondes. Pendant les accès mémoires non utiles, MRDY n'a pas d'effet sur l'allongement de E. Ceci évite le ralentissement de la vitesse du processeur pendant les accès bus non utiles.

– une ligne **DMA/BREQ** (Direct Memory Access/Bus Request = Accès Direct à la Mémoire/Demande de Bus) : l'entrée **DMA/BREQ** offre une méthode de suspension d'exécution et d'acquisition du bus du microprocesseur

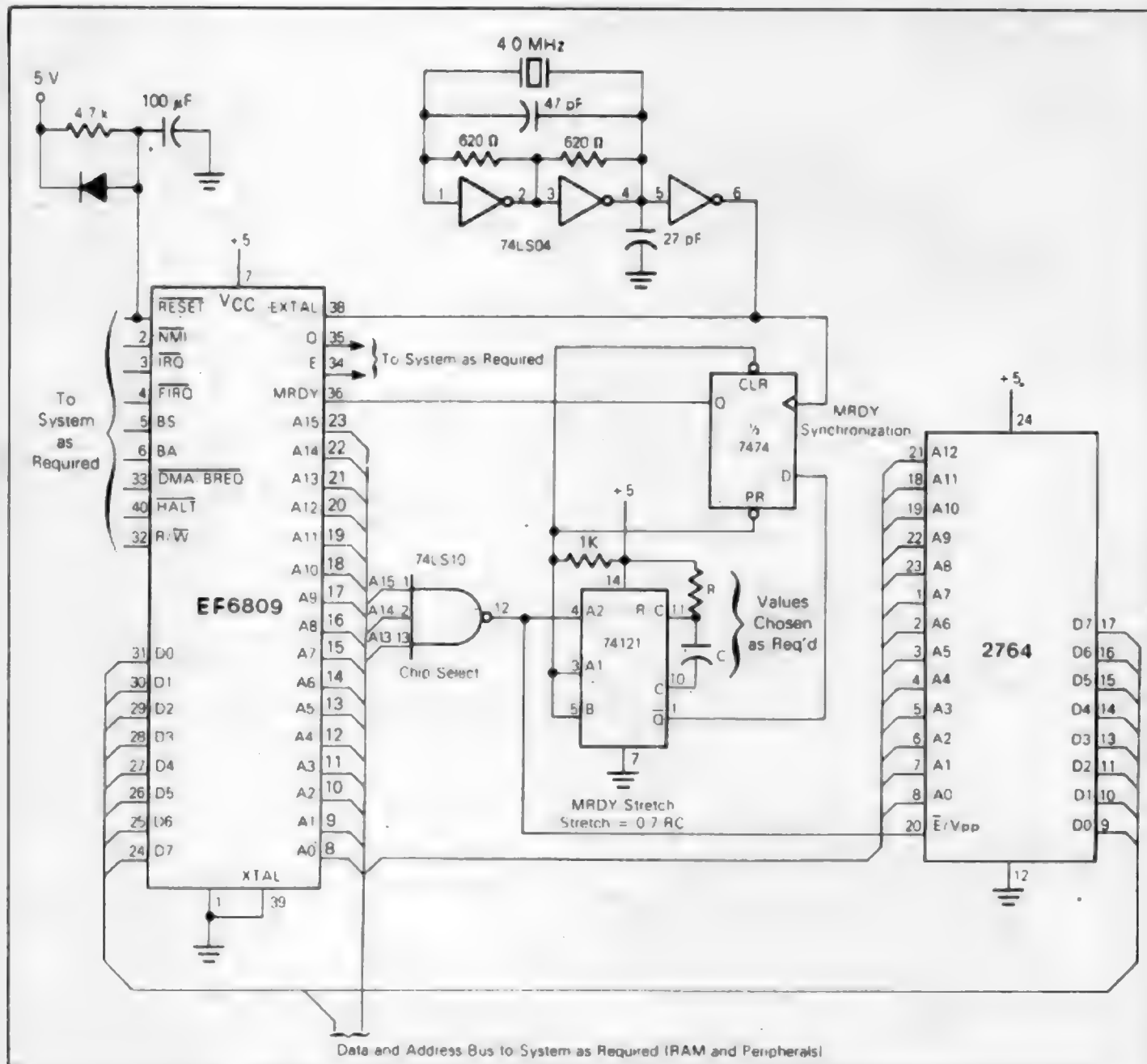


Fig 56b : MRDY synchronisation.

## VALEURS LIMITES

Valeurs	Symboles	Valeurs	Unités
Tension d'alimentation	V <sub>CC</sub>	-0.3 à +7.0	V
Tension d'entrée	V <sub>in</sub>	-0.3 à +7.0	V
Température de fonctionnement	T <sub>A</sub>	0 à +70	°C
Température de stockage	T <sub>stg</sub>	-55 à +150	°C
Résistance thermique	θ <sub>JA</sub>	70	°C/W

Les entrées de ce circuit sont protégées contre les hautes tensions statiques et les champs électriques ; toutefois, il est recommandé de prendre les précautions normales pour éviter toute tension supérieure aux valeurs limites sur ce circuit à haute impédance.

## CARACTÉRISTIQUES ÉLECTRIQUES (V<sub>CC</sub> = 5.0 V ± 5 %, V<sub>SS</sub> = 0, T<sub>A</sub> = 0 à 70°C sauf spécifications contraires)

Caractéristiques	Symboles	Min	Typ	Max	Unités
Tension d'entrée à l'état haut logique, E <sub>Xtal</sub> RESET	V <sub>IH</sub>	V <sub>SS</sub> + 2.0 V <sub>SS</sub> + 4.0	—	V <sub>DD</sub> V <sub>DD</sub>	V
Tension d'entrée à l'état bas logique, RESET, E <sub>Xtal</sub>	V <sub>IL</sub>	V <sub>SS</sub> - 0.3	—	V <sub>SS</sub> + 0.8	V
Courant de fuite en entrée (V <sub>in</sub> = 0 à 5.25 V, V <sub>CC</sub> = max)	I <sub>in</sub>	—	1.0	2.5	μA
Tension de sortie à l'état haut I <sub>charge</sub> = -205 μA, V <sub>CC</sub> = min) I <sub>charge</sub> = -145 μA, V <sub>CC</sub> = min) I <sub>charge</sub> = -100 μA, V <sub>CC</sub> = min)	V <sub>OH</sub>	V <sub>SS</sub> + 2.4 V <sub>SS</sub> + 2.4 V <sub>SS</sub> + 2.4	— — —	— — —	V
Tension de sortie à l'état bas (I <sub>charge</sub> = 20 mA, V <sub>CC</sub> = min)	V <sub>OL</sub>	—	—	V <sub>SS</sub> + 0.5	V
Puissance dissipée	P <sub>D</sub>	—	—	1.0	W
Capacité $\pi$ (V <sub>in</sub> = 0, T <sub>A</sub> = 25°C, f = 1.0 MHz)	C <sub>in</sub>	—	10 7	15 10	pF
	C <sub>out</sub>	—	—	12	
Fréquence de travail (Quartz ou entrée extérieure)	1 f <sub>XTAL</sub> f <sub>XTAL</sub>	— — —	— — —	4 6 8	MHz
Trois états, courant d'entrée (V <sub>in</sub> = 0.4 à 2.4 V, V <sub>CC</sub> = max)	I <sub>TSI</sub>	—	2.0	10 100	μA

## CARACTÉRISTIQUES DYNAMIQUES (lecture - écriture) (figures 1 et 2)

Caractéristiques	Symboles	EF6809			EF68A09			EF68B09			Unités
		Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
Temps de cycle	t <sub>CYC</sub>	1000	—	—	667	—	—	500	—	—	ns
Temps utile total	t <sub>UT</sub>	975	—	—	640	—	—	480	—	—	ns
Temps d'accès à la lecture t <sub>ac</sub> = (t <sub>AD</sub> - t <sub>DSR</sub> )	t <sub>ACC</sub>	695	—	—	440	—	—	320	—	—	ns
Temps d'étab. données (lect.)	t <sub>DSR</sub>	80	—	—	60	—	—	40	—	—	ns
Temps de maintien donn. (ent.)	t <sub>DHR</sub>	10	—	—	10	—	—	10	—	—	ns
Temps de maintien donn. (sort.)	t <sub>DHW</sub>	30	—	—	30	—	—	30	—	—	ns
Temps maintien des adresses (adresses, R/W)	t <sub>AH</sub>	30	—	—	30	—	—	30	—	—	ns
Temps de retard des adresses	t <sub>AD</sub>	—	—	200	—	—	140	—	—	110	ns
Temps de retard des données	t <sub>DDW</sub>	—	—	225	—	—	180	—	—	145	ns
Temps entre E bas et Q haut	t <sub>AVS</sub>	—	—	250	—	—	165	—	—	125	ns
Temps entre adresse et Q haut	t <sub>AO</sub>	25	—	—	25	—	—	15	—	—	ns
Horloge proces., état bas	t <sub>PWEL</sub>	450	—	—	295	—	—	210	—	—	ns
Horloge proces., état haut	t <sub>PWEH</sub>	450	—	—	280	—	—	220	—	—	ns
MRDY, temps d'établissement	t <sub>PCSR</sub>	60	—	—	60	—	—	60	—	—	ns
Temps d'établis., interruptions	t <sub>PCS</sub>	200	—	—	140	—	—	110	—	—	ns
HALT, temps d'établissement	t <sub>PCSH</sub>	200	—	—	140	—	—	110	—	—	ns
RESET, temps d'établissement	t <sub>PCSR</sub>	200	—	—	140	—	—	110	—	—	ns
DMA/BREQ, temps d'établis.	t <sub>PCSD</sub>	125	—	—	125	—	—	125	—	—	ns
Temps démarrage de l'oscillateur	t <sub>rc</sub>	100	—	—	100	—	—	100	—	—	ms
E, tps de montée et de descente	t <sub>ER</sub> t <sub>EF</sub>	5	—	25	5	—	25	5	—	20	ns
Temps de montée et de descente du processeur	t <sub>PCR</sub> t <sub>PLF</sub>	—	—	100	—	—	100	—	—	100	ns
Q, tps de montée et de descente	t <sub>QR</sub> t <sub>QF</sub>	5	—	25	5	—	25	5	—	20	ns
Q, état haut	t <sub>PWQH</sub>	450	—	—	280	—	—	220	—	—	ns

pour une autre utilisation comme montré en figure 57. Des utilisations types comprennent le DMA et le rafraichissement des mémoires dynamiques.

La transition de  $\overline{\text{DMA/BREQ}}$  doit se produire pendant Q. Un niveau bas sur cette broche arrêtera l'exécution de l'instruction à la fin du cycle en cours.

$\text{BA} = \text{BS} = 1$  indique la prise en compte de la demande faite par  $\overline{\text{DMA/BREQ}}$ , le circuit demandeur aura alors jusqu'à 15 cycles bus avant que le microprocesseur ne récupère le bus pour auto-rafraichissement.

L'auto-rafraichissement nécessite un cycle bus comportant un cycle perdu de début et de fin (figure 57).

En général, le contrôleur de DMA fait une demande d'accès au bus en mettant au niveau bas la broche  $\overline{\text{DMA/BREQ}}$  sur le front montant de E. Lorsque le microprocesseur répond avec  $\text{BA} = \text{BS} = 1$ , ce cycle est un cycle perdu utilisé pour transférer le contrôle au système de DMA.

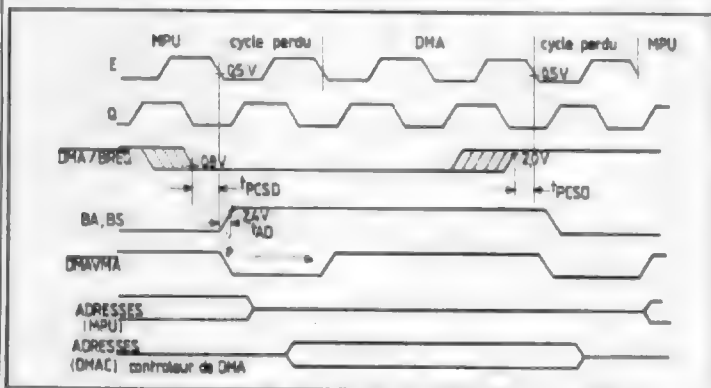


Fig. 57a : Diagramme de temps typique de l'entrée DMA ( $< 14$  cycles).

Fig. 57.  $\overline{\text{DMA/VMA}}$  est un signal externe au CPU, mais nécessaire pour la gestion du DMA.

Les faux accès mémoires doivent être évités pendant tout cycle perdu. Lorsque  $\text{BA}$  est remis à 0 (soit comme résultat de  $\overline{\text{DMA/BREQ}} = 1$ , ou auto-rafraichissement du microprocesseur), le circuit DMA doit être déconnecté du bus. Un autre cycle perdu s'écoule avant que le microprocesseur ne se voit allouer un accès mémoire pour transférer le contrôle sans litige.

**Nota :** se reporter au chapitre III pour de plus amples informations sur les lignes  $\overline{\text{RESET}}$ ,  $\text{NMI}$ ,  $\overline{\text{FIRQ}}$ ,  $\overline{\text{IRQ}}$ ,  $\overline{\text{Halt}}$ ,  $\overline{\text{DMA/BREQ}}$ .

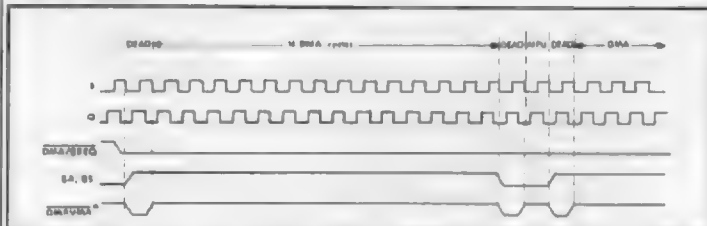


Fig. 57b : Diagramme des temps auto-rafraichissement en DMA ( $> 14$  cycles).

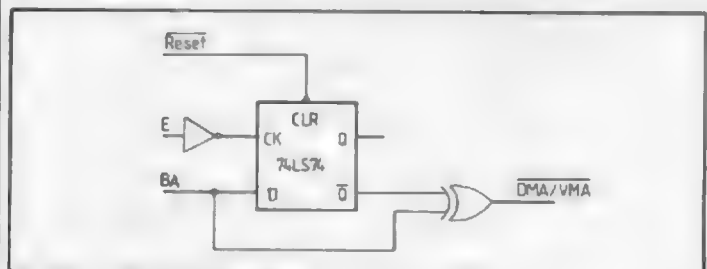


Fig. 57c : Logique de  $\overline{\text{DMA/VMA}}$ .



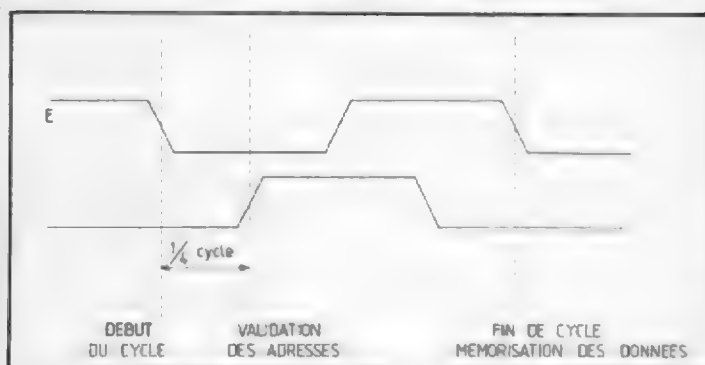
# Un petit système d'initiation : le Microkit 09

$R/\bar{W} = 1$  — lecture, le bus de données est en entrée  
 $R/\bar{W} = 0$  — écriture, le bus de données est en sortie

— **E et Q (broches 34 et 35)** : signaux d'horloge du CPU.  
**E** : signal de synchronisation avec la périphérie. La fréquence de cette ligne est celle de base du microprocesseur (horloge divisée par 4).

**Q** : ce signal est en quadrature avec E.

Les adresses des microprocesseurs sont correctes à partir d'un front montant de Q. Les données sont mémorisées sur un front descendant de E. Ces deux lignes sont des sorties.



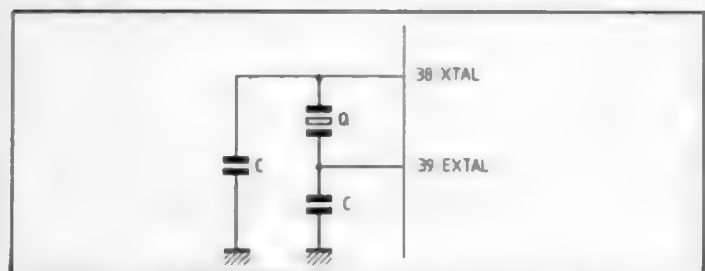
— **Halt (broche 40)** : cette ligne permet d'interrompre un programme en cours en appliquant un niveau bas sur cette entrée.

Le microprocesseur termine l'instruction en cours puis arrête le déroulement du programme.

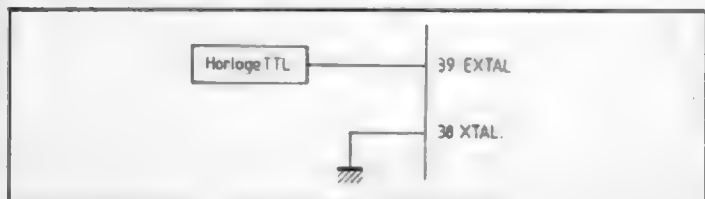
Lorsque cette ligne repasse au niveau haut, le programme se continue sans qu'il y ait perte d'information.

— **XTAL et EXTAL (broches 38 et 39)** : ces broches sont connectées au circuit horloge du CPU. Deux schémas peuvent être adoptés pour ces entrées :

— Résonance parallèle :



— Possibilité de brancher une horloge TTL externe :



Le quartz ou l'horloge externe sont égales à quatre fois la fréquence bus (E et Q). Le schéma choisi pour le micro-kit est le premier avec  $Q = 4 \text{ MHz}$  et  $C = 22 \text{ pF}$ .

— **RESET (broche 37)** : broche d'initialisation Hard du microprocesseur. Cette ligne active au niveau bas a pour effets :

- arrêt de l'instruction en cours
- registre de page DP = 00
- interruptions  $\bar{IRQ}$  et  $\bar{FIRQ}$  sont masquées
- l'interruption non masquable  $\bar{NMI}$  est désarmée

Le vecteur d'initialisation est disponible aux adresses FFFE et FFFF, c'est-à-dire que le microprocesseur dépose dans un premier temps l'adresse FFFE sur le bus d'adresses et lit un premier octet que nous appellerons octA, puis dépose FFFF et lit un second octet que nous appellerons octB. L'adresse formée par octA et octB est celle de la première instruction exécutée par le microprocesseur.

Cette ligne est une entrée.

(Toute mise sous tension commence automatiquement par un RESET.)

— **MRDY (broche 36)** : cette broche permet par un allongement de l'horloge E du CPU, d'autoriser la lecture ou l'écriture d'une mémoire ou d'un périphérique dont le temps d'accès est plus lent que celui du 6809; La valeur maximale est de  $10 \mu\text{s}$ . Cette ligne est une entrée.

— **DMA/BREQ (broche 33)** : cette ligne permet une utilisation des bus du microprocesseur par un circuit extérieur (mise en haute impédance des bus). Par exemple, pour faire du DMA ou du rafraîchissement mémoire.

Ainsi s'achève la description des broches du microprocesseur.

### Brochages des mémoires de la carte CPU

Les mémoires utilisées sont les suivantes :

— mémoire vive : RAM 6116 qui est une ram de  $2 \text{ K} \times 8$  contenant chacune un mot de 8 bits.

(Random Access Memory) = RAM

$2048 \text{ adresses} = 2\text{K}$

— mémoire morte : EPROM 2716 qui est une EPROM de  $2 \text{ K} \times 8$ .

(Erasable program read only memory) = EPROM.

Le brochage de ces deux circuits sera étudié en même temps du fait que seules quelques broches ont des fonctions différentes.

Le brochage de ces deux circuits est donné à la figure 2.

— **A0 à A10** : bus d'adresses des mémoires. Ces lignes sont connectées directement sur le bus d'adresses du CPU. Ce sont des entrées.

— **D0 à D7** : bus de données des mémoires. Ces lignes sont connectées directement sur le bus de données du 6809 ; ces lignes sont bidirectionnelles pour la RAM 6116 et à lecture seule pour la ROM 2716 (sorties).

— **Vcc et Vss (broches 24 et 12)** : lignes d'alimentation des mémoires  $V_{cc} = +5 \text{ V}$  et  $V_{ss} = \text{masse}$ .

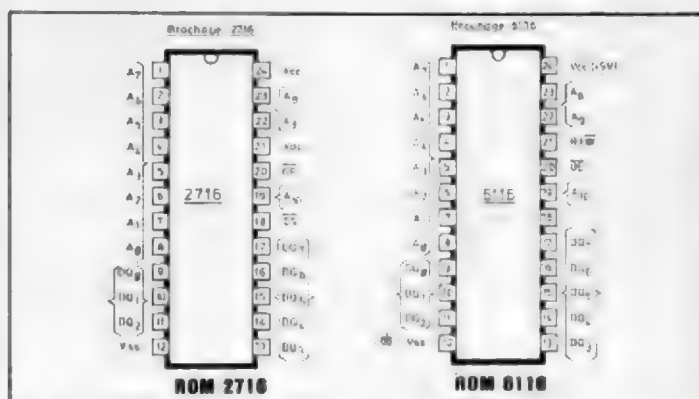


Fig. 2

-  $\overline{CS}$  (broches 18) ou Chip Select : broche de sélection du boîtier. Un niveau bas sur cette entrée signale au circuit considéré que le microprocesseur désire dialoguer avec lui. Cette broche a pour effet de passer le bus de données des mémoires dans un état haute impédance lorsqu'elle est au niveau 1 logique (+5 V).

-  $\overline{OE}$  (broche 20) ou Output Enable : lorsqu'elle est à 1, cette ligne permet une diminution de la consommation du boîtier pour alimenter, par exemple, le boîtier avec une pile (sauvegarde de la RAM alimentation coupée). Au niveau bas, cette ligne a le même effet que le CS.

Pour sélectionner un boîtier, il faut donc que les broches  $\overline{CS}$  et  $\overline{OE}$  soient toutes les deux au 0 logique (voir décodage d'adresse).

-  $R/\overline{W}$  (broche 21 de 6116) : ligne d'écriture ou de lecture de la RAM. Cette ligne donne le sens des données :

$R/\overline{W} = 1 \rightarrow D0$  à  $D7$  de la RAM en sortie.

$R/\overline{W} = 0 \rightarrow D0$  à  $D7$  de la RAM en entrée.

La ROM 2716 ne possède pas cette ligne du fait qu'elle est à lecture seule.

-  $V_{pp}$  (broche 21 de la 2716) : cette ligne est utilisée pour programmer la mémoire 2716. En mode normal, cette ligne est au +5 V. En mode programme, cette ligne est à +25 V. Pour le Micro Kit cette ligne est reliée en permanence au +5 V.

### Circuit de décodage d'adresse de la carte CPU (74LS138)

Ce circuit est un multiplexeur/démultiplexeur 1 parmi 8 dont le brochage est donné figure 3.

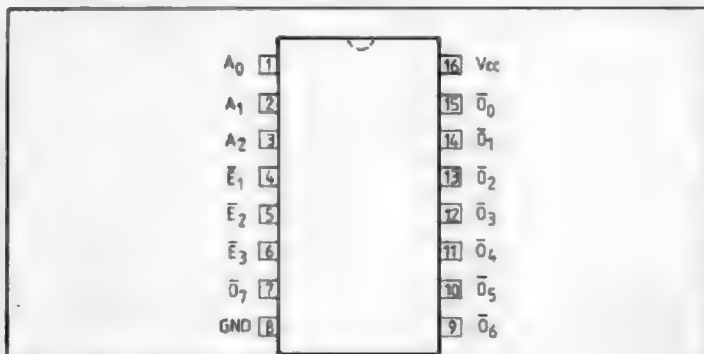


Fig. 3 : 74LS 138.

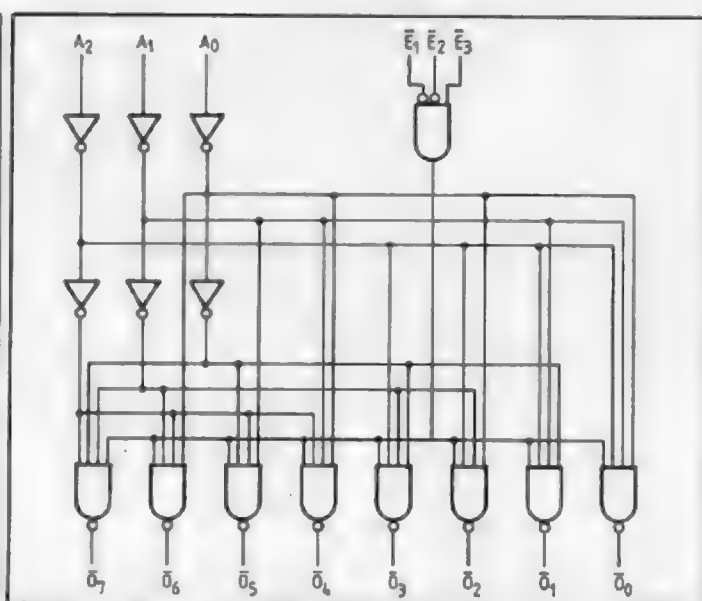


Schéma de principe du 74LS 138.

Piutôt qu'une description des broches, la table de vérité ci-dessous est plus parlante et évite un long discours.

TRUTH TABLE

INPUTS						OUTPUTS							
$\overline{E}_1$	$\overline{E}_2$	$\overline{E}_3$	$A_0$	$A_1$	$A_2$	$\overline{O}_0$	$\overline{O}_1$	$\overline{O}_2$	$\overline{O}_3$	$\overline{O}_4$	$\overline{O}_5$	$\overline{O}_6$	$\overline{O}_7$
H	X	X	X	X	X	H	H	H	H	H	H	H	H
X	H	X	X	X	X	H	H	H	H	H	H	H	H
X	X	L	X	X	X	H	H	H	H	H	H	H	H
L	L	H	L	L	L	L	H	H	H	H	H	H	H
L	L	H	H	L	L	H	L	H	H	H	H	H	H
L	L	H	L	H	L	H	H	L	H	H	H	H	H
L	L	H	H	H	L	H	H	H	L	H	H	H	H
L	L	H	L	L	H	H	H	H	H	L	H	H	H
L	L	H	H	L	H	H	H	H	H	H	L	H	H
L	L	H	L	H	H	H	H	H	H	H	H	L	H
L	L	H	H	H	H	H	H	H	H	H	H	H	L

H = HIGH Voltage Level L = LOW Voltage Level X = Immatériel

### Brochage du PIA 6821

Ce circuit est un périphérique d'entrée-sortie parallèle permettant sur le Micro Kit 09 de gérer toute la carte clavier (afficheur, clavier, etc...).

Le brochage de ce circuit est donné en figure 5.

-  $V_{cc}$  et  $V_{ss}$  (broches 1 et 20) : lignes d'alimentation du 6821.

$V_{ss}$  = masse et  $V_{cc}$  = +5 V

La fonction de ce circuit dans le Micro Kit est de découper l'espace mémoire adressable par le 6809 en 8 blocs de 8 Koctets (8192 adresses) afin de fabriquer avec le bus d'adresses du 6809 les signaux CS des différents circuits présents sur la carte CPU.

Le découpage mémoire obtenu avec ce circuit est donné en figure 4.



- $\overline{\text{RESET}}$  (broche 34) : ligne d'initialisation du périphérique généralement connectée à la ligne  $\overline{\text{RESET}}$  du CPU.
  - $\overline{\text{IRQA}}$  et  $\overline{\text{IRQB}}$  (broches 37 et 38) : lignes d'interruption du périphérique correspondant respectivement au port A et au port B.
  - $\text{CS0}$ ,  $\overline{\text{CS2}}$  et  $\text{CS1}$  (broches 22 à 24) ou Chip Select : lignes de sélection du boîtier.  $\text{CS0}$  et  $\text{CS1}$  sont actives au niveau 1 logique alors que  $\overline{\text{CS2}}$  est active au niveau bas.
- Ainsi s'achève la description des circuits de la carte CPU. Nous allons maintenant pouvoir entrer dans le vif du sujet.

Cette carte est composée du clavier évidemment, mais aussi des afficheurs, permettant de visualiser toutes les informations nécessaires, ainsi que les circuits de commande des afficheurs et l'interface cassette.

Ce circuit est un octal Buffer couramment utilisé en microinformatique. Le brochage est donné figure 6.



C'est un multiplexeur démultiplexeur 1 parmi 10. Table de vérité et brochage en figure 7.

TRUTH TABLE

H = HIGH Voltage Level  
L = LOW Voltage Level



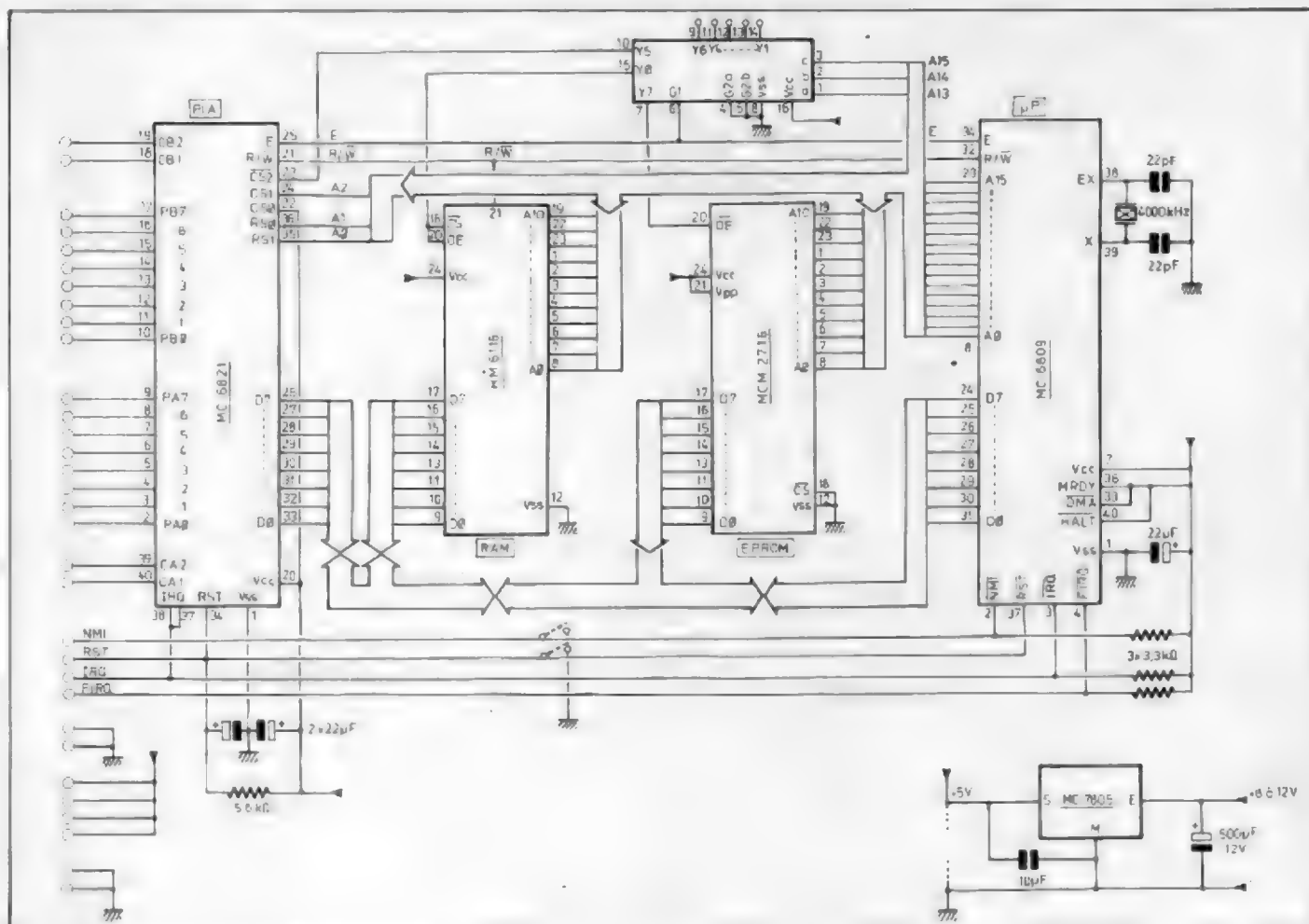


Fig. 8 : Schéma de la carte CPV.

## MONTAGE DE LA MAQUETTE

Si le câblage des deux cartes du Micro Kit 09 ne présente pas vraiment de grosses difficultés, il n'en est pas de même pour la gravure des deux circuits imprimés en double face. Afin d'aider au maximum les lecteurs intéressés par cette réalisation, nous allons mettre à leur disposition le jeu de circuits, ceux-ci étant réalisés à trous métallisés afin de simplifier les interconnexions entre les deux faces.

### Construction de la maquette

Voici quelques indications pour faciliter le montage de la maquette. Pour le montage des composants, s'aider des schémas des C.I. (fig. 11 à 13).

- (1) Montage de la carte centrale (petite carte). Voir figures 8 et 13.
  - Repérer la face composants (comportant l'inscription MICROKIT 09).
  - Placer et souder les supports 40, 24 et 16 broches pour les circuits intégrés. Attention au sens (repère).
  - Placer et souder les supports spéciaux d'interconnexion des cartes centrale et périphérique.
  - Placer et souder résistances et condensateurs. Attention

à la polarité des condensateurs électrolytiques.

- Placer et souder le quartz. (Ne pas trop enfoncer les pattes pour ne pas court-circuiter le boîtier avec une piste).
- Placer et souder les picots et fils d'alimentation ou le régulateur de tension et son radiateur.

- (2) Montage de la carte périphérique (grande carte). Voir figures 9 et 12.

- Repérer la face touches (elle comporte l'inscription MICROKIT 09)
- Placer et souder les bases de touches.
- Préparer éventuellement les six supports 14 broches des afficheurs en coupant les pattes inutiles. Les placer et les souder.
- Couper et placer les barettes sécables à wrapper servant à l'interconnexion des cartes (2x14 et 2x16 broches). Attention à ne pas faire couler de la soudure sur les pattes à wrapper.
- Placer et souder l'inverseur de l'interface cassette.

Retourner la carte et câbler le dessous :

- Placer et souder les résistances
- Placer et souder les supports 20, 16 et 8 broches. Res-



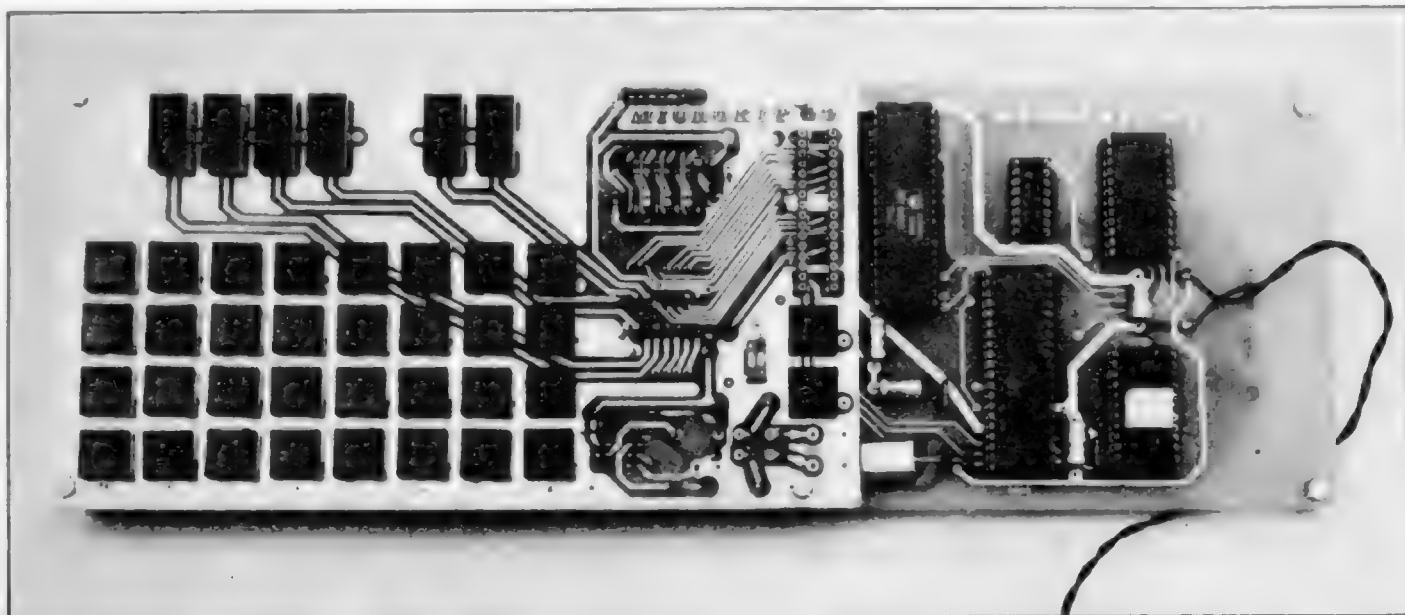
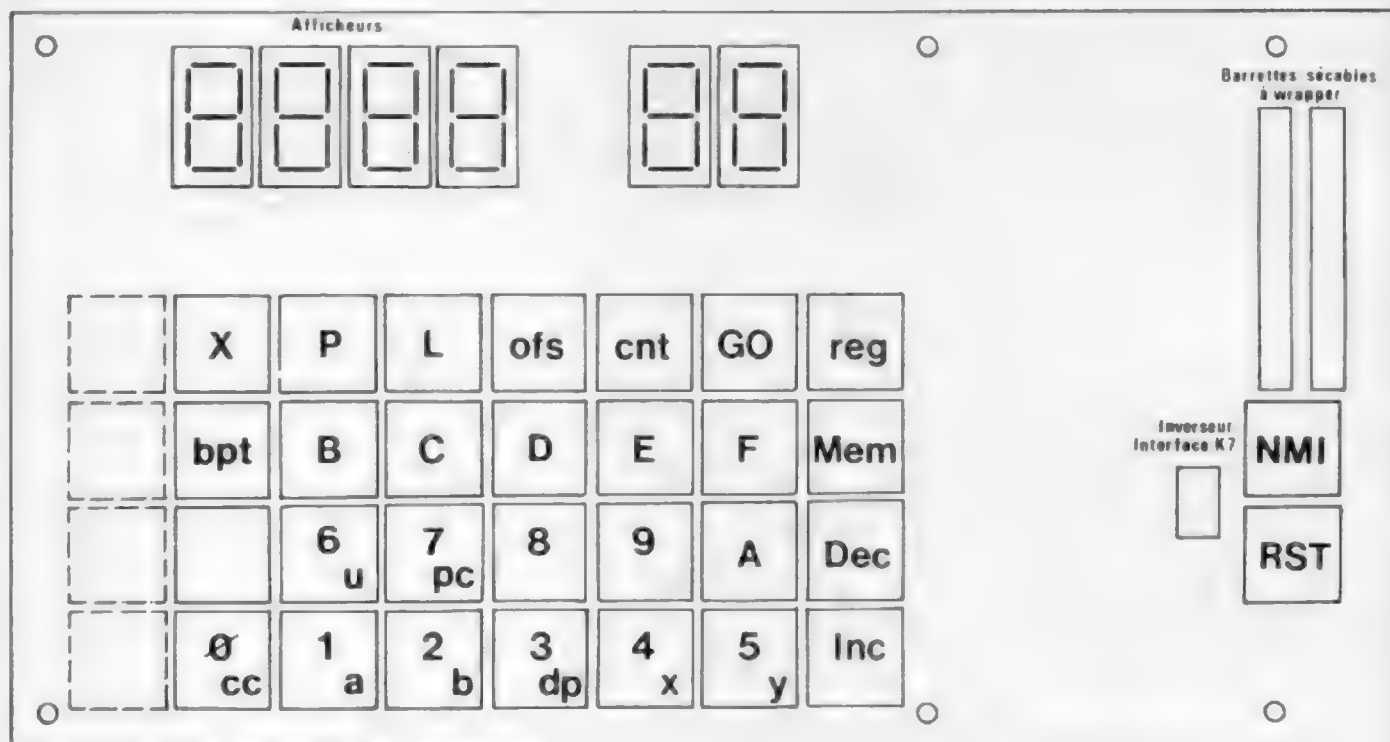


Fig. 10 : Organisation sérigraphique de la carte Clavier.

INTERCONNEXION AVEC CARTE PERIPHERIQUE

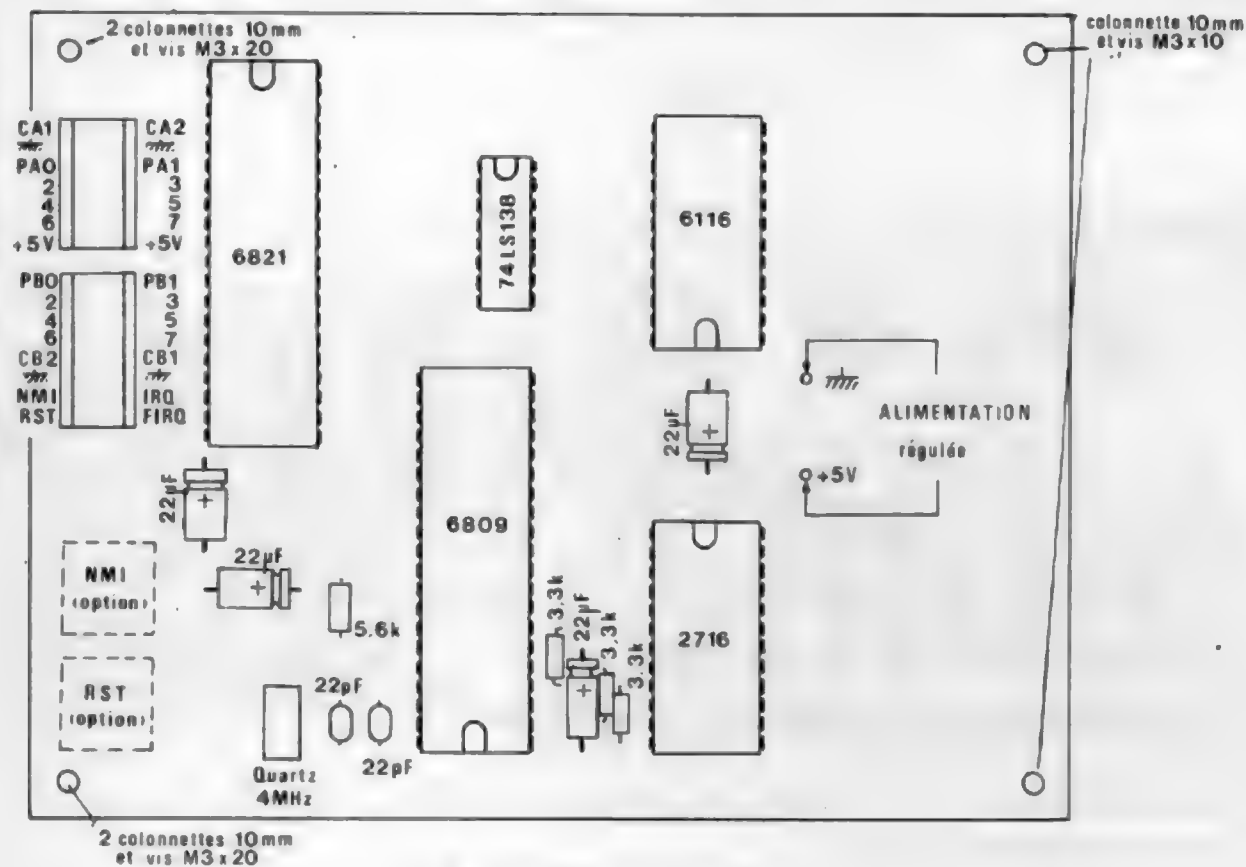


Fig. 11 : Implantation des composants.



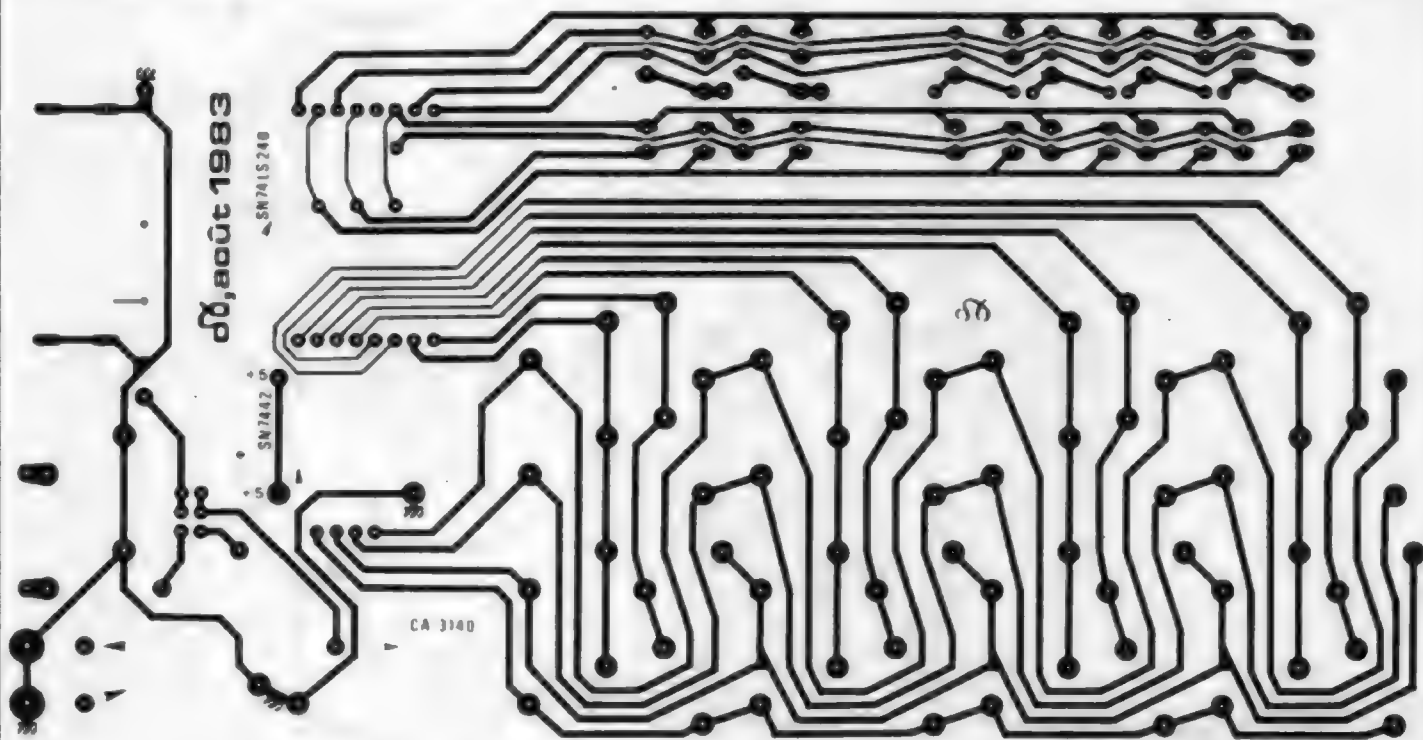
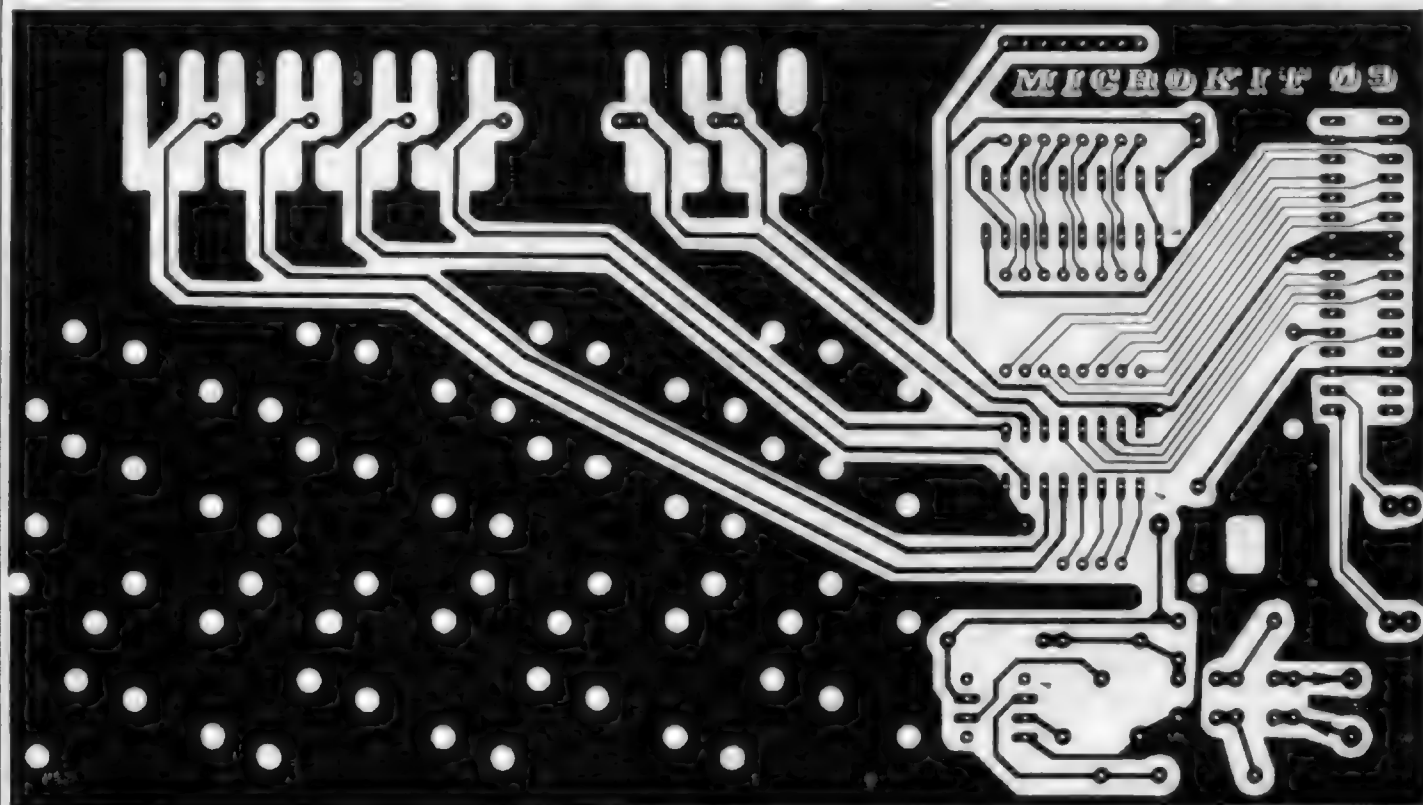


Fig. 12 : Carte Clavier.

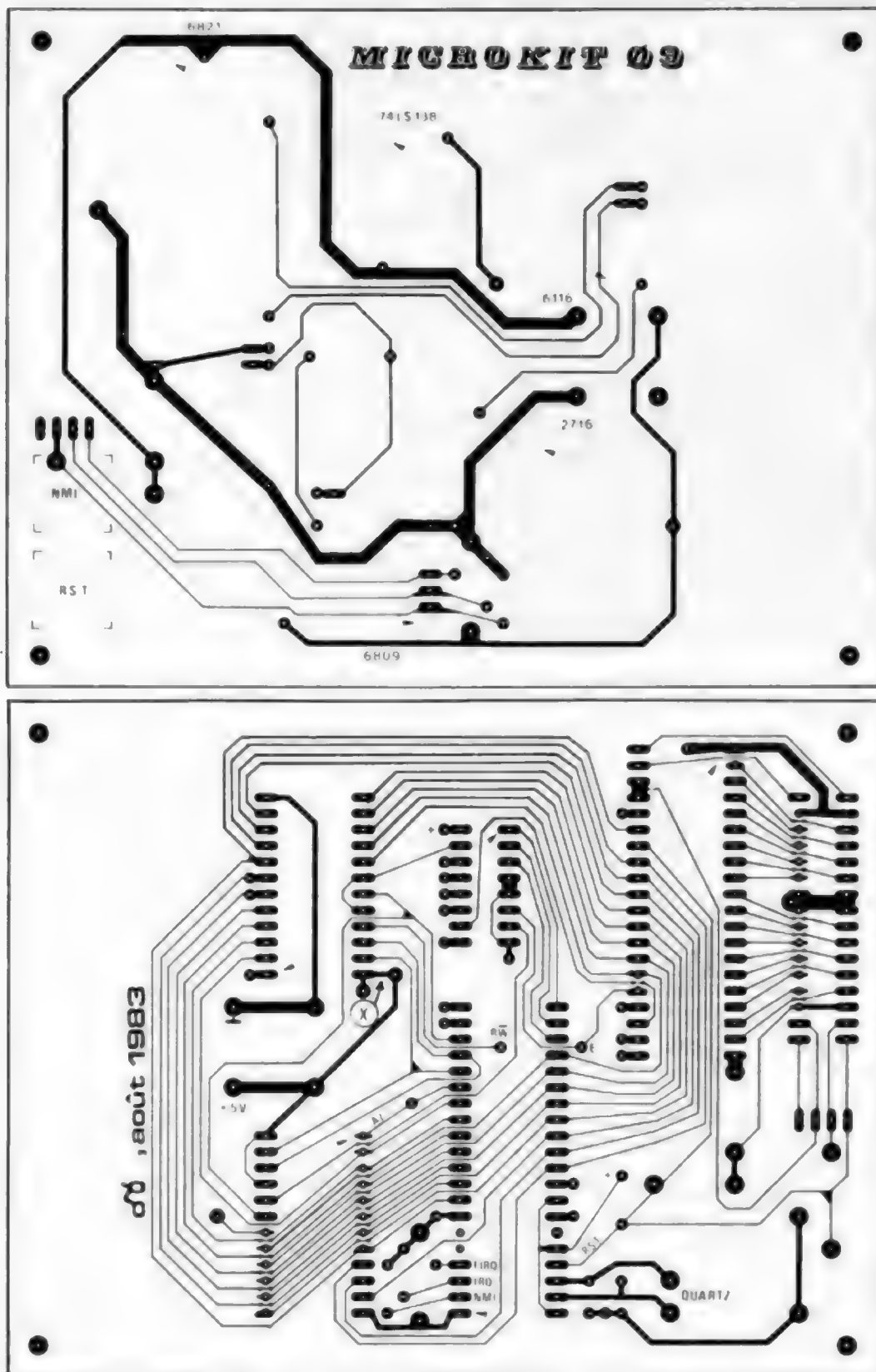


Fig. 13 : Carte Unité centrale.

## L'ARCHITECTURE INTERNE DU 6809

L'architecture interne du 6809 est orientée 16 bits, ce qui fait de lui l'un des plus puissants microprocesseurs 8 bits du marché. En effet, ce microprocesseur possède deux accumulateurs 8 bits appelés A et B pouvant être associés pour former un seul registre 16 bits appelé D. Fig. 14.

Il possède également deux registres index X et Y (16 bits), deux registres pointeurs de piles U et S (16 bits) pouvant servir d'index, un pointeur de page DP (8 bits) permettant de découper l'espace mémoire adressable en 256 pages de 256 octets chacune et enfin un registre d'état de 8 bits CC. La figure ci-dessous donne le détail de l'architecture du 6809.

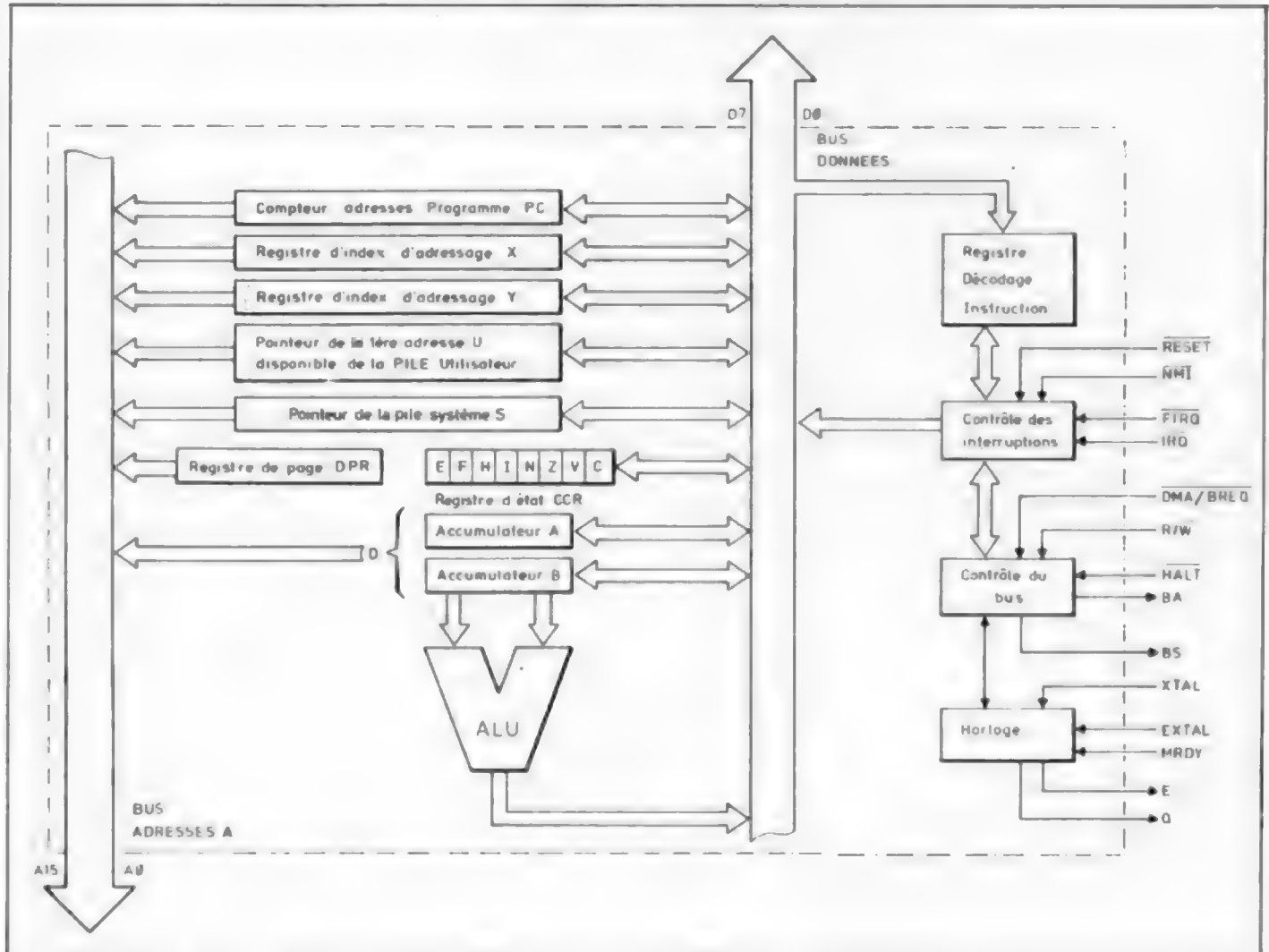


Fig. 14 : Architecture du 6809.

## PROGRAMMATION - UTILISATION DE LA MAQUETTE

Maintenant que votre maquette fonctionne, il est temps de l'utiliser. Pour cela nous allons passer en revue toutes les touches de fonction du clavier.

Comme nous l'avons dit précédemment, à la mise sous tension, la barre centrale de l'afficheur de gauche doit s'allumer indiquant que le système est sous contrôle du moniteur et qu'il attend l'enfoncement d'une touche.

### Touches de fonction

- Mem (mémoire) : cette touche permet la lecture et la modification de la case mémoire de votre choix. Dès l'appui sur cette touche, le symbole  $\Pi$  doit apparaître sur l'afficheur de droite.

Le numéro de la mémoire peut alors être entré, et il se visualisera sur les quatre afficheurs de gauche.

Une fois l'adresse entrée, le contenu de cette adresse est visualisé sur les afficheurs de droite.

Dans le cas où vous voudriez modifier le contenu de cette

case mémoire, il suffit d'entrer la nouvelle donnée par l'appui successif sur deux des 16 touches hexadécimales. Notons qu'un premier appui modifie le chiffre des poids faibles. Lors du second appui, ce chiffre est transféré à gauche sur l'afficheur des poids forts pour permettre d'entrer le chiffre des poids faibles. La nouvelle valeur entrée au clavier est mémorisée en RAM (si bien entendu, l'adresse spécifiée se trouve en RAM).

Si par hasard, vous avez fait une erreur de frappe, continuez à taper les bonnes valeurs sur les touches hexadécimales. La donnée mémorisée sera toujours la dernière entrée.

**Inc**: touche «Increment». Permet d'incrémenter (= augmenter de 1 l'adresse lors de la visualisation du contenu d'une case mémoire. Cette touche est couramment utilisée lors de la rentrée d'un programme utilisateur en mémoire RAM. Cette touche permet aussi la visualisation des registres l'un après l'autre. Ce mode visualisation des registres peut soit être consécutif à une instruction d'interruption logicielle (SWI : code 3F) rencontrée lors du déroulement d'un programme utilisateur, soit être appelé par l'introduction d'un point d'arrêt dans le programme utilisateur.

**Dex**: touche «Décrement». Cette touche a le même rôle que la précédente mais au lieu d'incrémenter l'adresse d'un pas, elle la décremente (= diminuer) d'un pas.

**Reg**: touche «Registre». Permet de visualiser le contenu des registres CCR, A, B, DP, X, Y, U, PC de l'unité centrale 6809 (fig. 4). Un appui sur la touche «reg» provoque l'affichage du symbole «Γ» sur le cinquième afficheur. La sélection du registre s'effectue alors grâce aux touches hexadécimales :

- 0 pour le registre CC
- 1 pour le registre A
- 2 pour le registre B
- 3 pour le registre DP
- 4 pour le registre X
- 5 pour le registre Y
- 6 pour le registre U
- 7 pour le registre PC

Le symbole du registre apparaît sur le sixième afficheur, alors que son contenu est affiché sur les deux ou quatre premiers afficheurs selon la configuration B ou 16 bits du registre examiné.

Adresse	Code instruction hexadécimal	Langage symbolique (mnémotechnique)	Commentaire
0100	12	NOP	Pas d'opération effectuée
0101	12	NOP	Interruption du programme. Rappel du moniteur
0102	3F	SWI	
0103	12	NOP	
0104	12	NOP	
0105	3F	SWI	

**GO**: touche de lancement d'un programme. Le programme se déroulera dès que l'adresse de départ aura été spécifiée à l'aide de l'appui sur quatre touches hexadécimales. Pour expliciter le rôle des différentes touches, dont le fonctionnement a déjà été présenté, nous conseillons le court programme en encadré ci-dessus.

Procédure à suivre :

RST → «→» sur le premier afficheur

Mem → «Π».

Entrer l'adresse 0100, n° de la case-mémoire de départ du programme → une donnée s'affiche.

Entrer 1 puis 2.

Inc → l'adresse passe à 0101.

Entrer 12.

Inc

Entrer 3F.

etc... jusqu'à l'adresse 0105. Lancer le programme par RST, GO 0100 ou par GO, GO 0100 si on effectuait auparavant l'examen d'une case-mémoire.

Résultat sur l'affichage

Le programme a été exécuté jusqu'à l'instruction d'interruption logicielle SWI, ce qui provoque la visualisation des registres de l'unité centrale. Le premier visualisé est le registre d'état CC, symbolisé par **C**. En appuyant sur Inc, on visualise le contenu de A, de nouveau sur Inc celui de B puis DP, puis X, symbolisé par **H** (quatre afficheurs de gauche pour le contenu 16 bits de X) puis Y puis U et enfin le compteur programme PC qui contient 0103, valeur de l'adresse pointée afin de poursuivre le déroulement du programme.

Appui sur Dex → on repasse au registre U.

Inc → PC

Inc — le symbole **Γ** apparaît. Il est dès lors possible soit de visualiser le contenu d'un registre particulier (revoir la touche reg), soit de revenir sous programme moniteur par appui sur Inc.

On peut s'entraîner à ces manipulations en recommençant l'opération à partir de GO0100.

**Cnt**: touche «continue». Elle permet de poursuivre le déroulement d'un programme, après que celui-ci eut été arrêté par une instruction SWI ou par un point d'arrêt. Relançons notre programme par RST GO 0100. Le compteur programme pointe 0103. Un premier appui sur cnt fait apparaître le symbole «→». Un deuxième appui sur cnt provoque la poursuite du programme jusqu'à la deuxième instruction d'interruption codée 3F. On vérifie en effet que le compteur-programme pointe alors l'adresse 0106.


bp : touche «break point» (point d'arrêt). Elle permet de placer un point d'arrêt dans un programme à l'adresse spécifiée par l'utilisateur.


Un premier appui sur bp provoque l'affichage de **BP**. Entrer ensuite, par exemple, l'adresse 0102. Dès l'entrée du dernier chiffre d'adresse, le système passe sous contrôle moniteur (symbole «→»). Relançons notre programme GO 0100... Après exécution, le programme s'est arrêté en 0102 que pointe le PC. La poursuite s'effectue par cnt. L'utilisation du point d'arrêt permet la mise au point d'un programme («debuging») en vérifiant les registres de l'unité centrale.



ofs : touche «offset». Elle permet le calcul automatique de la valeur du déplacement lors de sauts ou de branchements relatifs dans un programme. Elle place la valeur hexadécimale de ce déplacement dans le programme utilisateur, situé en RAM. Ecrivons le petit programme, en encadré ci-dessous, afin de s'exercer à l'utilisation de cette touche : On écrit ce programme à partir de l'adresse 0200. On y entre B6, puis 00, ou encore 00 et enfin 27 à l'adresse 0203. L'instruction de branchement BEQ «branch if equal» comprend le code opération 27 qui doit être suivi de la valeur du déplacement exprimé en hexa (en code complément à 2/). Le calcul de cette valeur est sur le Microkit 09 automatique. Sans incrémenter (on est toujours à l'adresse 0203, donnée 27) appuyer sur ofs → donnera le symbole «R». Entrer par le clavier l'adresse de destination du branchement 0220 puis GO →. Le système place alors la valeur 1B (valeur en code complément à 2 du déplacement), à l'adresse 0204. Appuyer sur Inc, et entrer 3F. Puis entrer 3P à l'adresse 0220. Lancer le programme GO 0200. Après l'exécution du programme, le PC pointe l'adresse 0206 ou 0221 selon le contenu de la case-mémoire 0000, contenu que l'on retrouve d'ailleurs dans A.

P : touche «Punch» = enregistrement. Cette touche permet de transférer une zone RAM vers un magnétophone à cassette, pour y stocker programmes et données. Un appui sur P entraîne l'apparition du symbole S (start). Entrer l'adresse de début de la zone à transférer. Dès l'entrée du quatrième chiffre de l'adresse, le symbole F (fin) apparaît. Il faut entrer l'adresse de fin de la zone mémoire à transférer. Dès l'entrée du quatrième chiffre de l'adresse de fin, la transmission vers la cassette s'effectue. Le temps de transmission sera fonction de la longueur du bloc à transmettre. En fin de transmission, le symbole F apparaît sur le premier afficheur. Exercices : calculer la vitesse approximative de transmission... sec/2 koctets).

 : touche «Load» = lecture. Elle permet de charger en mémoire RAM un programme provenant d'un magnétophone à cassette.

 : cette touche combinée avec ofs permet le calcul automatique de déplacement en adressage indexé.

## LES INSTRUCTIONS DU 6809

Le 6809 possède un grand jeu d'instructions donné dans le tableau 1, lequel nous est fourni par le fabricant (MOTOROLA). Toutes ces instructions peuvent être classées en plusieurs catégories distinctes.

### Les instructions arithmétiques



ADD : addition du contenu d'une case mémoire avec un accumulateur  
 ADC : addition du contenu d'une case mémoire avec un accumulateur plus la retenue  
 ABX : addition du registre B avec le registre X  
 DAA : ajustement décimal de l'accumulateur A  
 MUL : multiplication de l'accumulateur A par l'accumulateur B (non signée)  
 SUB : soustraction du contenu d'une case mémoire à l'accumulateur  
 SBC : soustraction du contenu d'une case mémoire à l'accumulateur moins la retenue  
 SEX : extension de signes de l'accumulateur B à l'accumulateur A.

### Les instructions de rotation et de décalage

ASR : décalage arithmétique à droite  
 LSL ou ASL : décalage logique ou arithmétique à gauche  
 LSR : décalage logique à droite  
 ROL : rotation à gauche  
 ROR : rotation à droite  
 Ces instructions fonctionnent avec les accumulateurs A et B ainsi qu'avec n'importe quelle case mémoire.

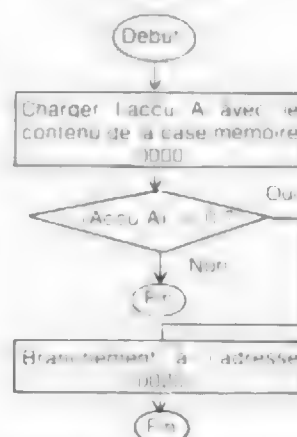
### Les instructions logiques

AND : «ET logique» entre une case mémoire et un registre interne (A, B, CC).  
 EOR : «OU exclusif» entre une case mémoire et un registre interne (A, B, CC).  
 OR : «OU logique» entre une case mémoire et un registre interne (A, B, CC).

Etiquette	Adresse	Code instruction Operation Operande	Mnemonique
	0200	B6 00 00	LDA > 0000
	0203	27 	BEQ END
	0205	3F 	SWI
	...	Valeur hexa du déplacement	
	0220	3F	SWI
END			

Programme permettant de s'exercer à l'utilisation de la touche OFS.

Ordinogramme



Mnémonique Instruction	Modes d'adressage du 6809																Indicateurs																		
	INHERENT				DIRECT				EXTENDED				IMMEDIAT				INDEXE				RELATIF														
	OP	N			OP	N			OP	N			OP	N			OP	N			OP	N			OP	N			DESCRIPTION	H	N	Z	V	C	
ABX	3A	3	1																										B + X - X	.	.	.	.	.	
ADCA				99	4	2		89	5	3		89	2	2			A9	4	2	+									A + M + C - A	.	.	.	.	.	
ADCB				D9	4	2		C9	5	3		C9	2	2			E9	4	2	+									B + M + C - B	.	.	.	.	.	
ADDA				9B	4	2		8B	5	3		8B	2	2			A9	4	2	+									A + M - A	.	.	.	.	.	
ADDB				DB	4	2		CB	5	3		CB	2	2			E9	4	2	+									B + M - B	.	.	.	.	.	
ADDD				D3	6	2		F3	7	3		C3	4	3			E3	6	2	+									D + M + 1 - D	.	.	.	.	.	
ANDA				94	4	2		B4	5	3		B4	2	2			A4	4	2	+									A M - A	.	.	.	.	.	
ANDB				D4	4	2		F4	5	3		C4	2	2			E4	4	2	+									B M - B	.	.	.	.	.	
ANDCC												1C	3	2															CC IMM - CC	.	.	.	.	1	
ASLA	48	2	1																										A	.	.	.	.	.	
ASLB	58	2	1																										B	.	.	.	.	.	
ASL				08	6	2		78	7	3							68	6	2	+									M	.	.	.	.	.	
ASRA	47	2	1																										A	.	.	.	.	.	
ASRB	57	2	1																										B	.	.	.	.	.	
ASR				07	6	2		77	7	3							67	6	2	-									M	.	.	.	.	.	
BCC																	24	3	2										Si C = 0	.	.	.	.	.	
LBCC																	10	5(6)	4															.	
																	24																		.
BCS																	25	3	2										Si C = 1	.	.	.	.	.	
LBCS																	10	5(6)	4															.	
																	25																		.
BEO																	27	3	2										Si Z = 1	.	.	.	.	.	
LBEO																	10	5(6)	4															.	
																	27																		.
BGE																	2C	3	2										Si > Zéro	.	.	.	.	.	
LBGE																	10	5(6)	4															.	
																	2C																		.
BGT																	2E	3	2										Si > Zéro	.	.	.	.	.	
LBGT																	10	5(6)	4															.	
																	2E																		.
BHI																	22	3	2																.
LBHI																	10	5(6)	4															.	
																	22																		.
BHS																	24	3	2																.
LBHS																	10	5(6)	4															.	
																	24																		.
BITA				95	4	2		85	5	3		85	2	2			A5	4	2	+										(M A A)	.	.	.	.	.
BITB				D5	4	2		F5	5	3		C5	2	2			E5	4	2	+										(M A B)	.	.	.	.	.
BLE																	2F	3	2										Si < Zéro	.	.	.	.	.	
LBLE																	10	5(6)	4															.	
																	2F																		.
BLO																	25	3	2																.
LBLO																	10	5(6)	4															.	
																	25																		.
BLS																	23	3	2																.
LBLS																	10	5(6)	4															.	
																	23																		.
BLT																	20	3	2										Si < Zéro	.	.	.	.	.	
LBLT																	10	5(6)	4															.	
																	20																		.
BMI																	2B	3	2																.
LBMI																	10	5(6)	4															.	
																	2B																		.
BNE																	26	3	2										Si Z = 0	.	.	.	.	.	
LBNE																	10	5(6)	4															.	
																	26																		.
BPL																	2A	3	2																.
LBPL																	10	5(6)	4															.	
																	2A																		.
BRA																	20	3	2																.
LBRA																	16	5	3																.

Tableau 1

codage mb de top d'horloge  
mb d'octets

OPÉRATION	Mnémonique Instruction	Modes d'adressage du 6809															DESCRIPTION	Indicateurs																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														
		INHERENT			DIRECT			EXTENDED			IMMÉDIAT			INDEXÉ				RELATIF																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																														
		OP	N		OP	N		OP	N		OP	N		OP	N			OP	N																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													
Non branchement	BRN LBRN																21 10 5 4																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																															

Tableau 1 (suite)

OPÉRATION	Mnémonique Instruction	Modes d'adressage du 6809															Indicateurs			
		INHERENT		DIRECT		EXTENDED		IMMÉDIAT		INDEXÉ		RELATIF				DESCRIPTION	5 3 2 1 0			
		OP	N	OP	N	OP	N	OP	N	OP	N	OP	N	OP	N		H	N	Z	V
Non opération	NOP	12	2	1																
• OU = logique mémoire et accumulateur	ORA			9A	4	2	BA	5	3	8A	2	2	AA	4	2	A ← A				
• OU = logique avec le registre codes condition	ORB			DA	4	2	FA	5	3	CA	2	2	EA	4	2	B ← B				
	OROC							1A	3	2						CC ← IMM - CC				7
Emplacement de tout(s) registre(s) (sauf S) sur la pile S	PSHS	34	5	2																
Emplacement de tout(s) registre(s) (sauf U) sur la pile U	PSHU	36	5	2																
Déploiement de tout(s) registre(s) de la pile S	PULS	35	5	2																
Déploiement de tout(s) registre(s) (sauf U) de la pile U	PULU	37	5	2																
Décalage circulaire à gauche du contenu mémoire ou de l'accumulateur	ROLA	49	2	1												A ← A				
	ROLB	59	2	1												B ← B				
	ROL				09	6	2	79	7	3				69	5	A ← A				
Décalage circulaire à droite du contenu mémoire ou de l'accumulateur	RORA	46	2	1												A ← A				
	RORB	56	2	1												B ← B				
	ROR				06	6	2	76	7	3				66	5	A ← A				
Retour d'interruption	RTI	3B	5	1																7
Retour de sous programme	RTS	39	5	1																
Soustraction du contenu mémoire de l'accumulateur	SBCA				97	4	2	B7	5	3	B2	2	2	A2	4	A ← A - M				
	SBCB				D7	4	2	F7	5	3	C2	2	2	E2	4	B ← B - M				
Extension de signe de l'accumulateur B à l'accumulateur A	SEX	1D	2	1																
Mise en mémoire du contenu de l'accumulateur	STA				97	4	2	B7	5	3				A7	4	A ← M				
	STB				D7	4	2	F7	5	3				E7	4	B ← M				
	STD				DD	5	2	FD	6	3				ED	5	D ← MM + 1				
	STS				10	6	3	70	7	4				10	6	S ← MM + 1				
Mise en mémoire du porteur de pile	STP				0F	5	2	FF	6	3				EF	5					
Mise en mémoire du registre index	STU				DF	5	2	FF	6	3				EF	5	U ← MM + 1				
	STX				9F	5	2	BF	6	3				AF	5	X ← MM + 1				
	STY				10	6	3	70	7	4				10	6	Y ← MM + 1				
					9F	5	2	BF	6	3				AF	5					
Soustraction du contenu mémoire de l'accumulateur	SUBA				90	4	2	B0	5	3	B0	2	2	A0	4	A ← A - M				
	SUBB				D0	4	2	F0	5	3	C0	2	2	E0	4	B ← B - M				
	SUBD				93	6	2	B3	7	3	B3	4	3	A3	6	D ← MM + 1 - D				
Interruption programmée	SWI	3F	19	1																
	SWI2	10	20	2																
	SWI3	11	20	2																
Synchronisation avec le signal d'interruption	SYNC	13	2	1																
Transfert de R1 à R2	TFR R1, R2	1F	7	2												R1 ← R2				
Test mémoire ou accumulateur	TSTA	4D	2	1																
	TSTB	5D	2	1																
	TST				0D	6	2	7D	7	3				6D	6					

Tableau 1 (suite)

## Légende

OP code hexadécimal de l'opération  
 ~ nombre de cycles - horloge  
 N nombre d'octets  
 + plus arithmétique  
 - moins arithmétique  
 X multiplication  
 M complément de M  
 - transfert dans  
 H demi-retenue du bit 3  
 N négatif (bit de signe)

Z Zéro (octet)  
 V Dépassement  
 C report du bit 7  
 I testé et mis à 1 si condition vraie, sinon mis à zéro  
 • non affecté  
 CC registre d'indicateur d'état  
 : concatenation  
 V OU logique  
 V OU logique Ex  
 A ET logique

SOURCE	DESTINATION
0000 : 0, A, B	0000 : A
0001 : 4	0001 : B
0010 : 7	0010 : CCR
0011 : 5	0011 : 3PR
0100 : 5	
0101 : PC	

Code registre

## Notes :

1. Dans le tableau sont donnés le nombre de cycles et d'octets de base. Pour déterminer le nombre total de cycles et d'octets ajouter les valeurs du tableau 2 des types d'adressage indexé.

2. R1 et R2 peuvent être une paire de registres 8 bits : A, B, CC, DP, ou 16 bits : X, Y, U, S, D, PC.

Il faut alors ajouter au code-instruction un post-octet selon le code-registre

Tableau 1 (suite)



Type	Forme	Non Indirect				Indirect			
		Assembler Form	Postbyte OP Code	+	-	Assembler Form	Postbyte OP Code	+	-
Constant Offset From R (2's Complement Offsets)	No Offset	.R	1RR00100	0	0	[.R]	1RR10100	3	0
	5 Bit Offset	n, R	0RRnnnnn	1	0	defaults to 8-bit			
	8 Bit Offset	n, R	1RR01000	1	1	[n, R]	1RR11000	4	1
	16 Bit Offset	n, R	1RR01001	4	2	[n, R]	1RR11001	7	2
Accumulator Offset From R (2's Complement Offsets)	A Register Offset	A, R	1RR00110	1	0	[A, R]	1RR10110	4	0
	B Register Offset	B, R	1RR00101	1	0	[B, R]	1RR10101	4	0
	D Register Offset	D, R	1RR01011	4	0	[D, R]	1RR11011	7	0
Auto-Increment/Decrement R	Increment By 1	.R +	1RR00000	2	0	not allowed			
	Increment By 2	.R + +	1RR00001	3	0	[.R + +]	1RR10001	6	0
	Decrement By 1	.- R	1RR00010	2	0	not allowed			
	Decrement By 2	.- - R	1RR00011	3	0	[.- - R]	1RR10011	6	0
Constant Offset From PC (2's Complement Offsets)	8 Bit Offset	n, PCR	1xx01100	1	1	[n, PCR]	1xx11100	4	1
	16 Bit Offset	n, PCR	1xx01101	5	2	[n, PCR]	1xx11101	8	2
Extended Indirect	16 Bit Address	-	-	-	-	[n]	10011111	5	2

R = X, Y, U or S  
 x = Don't Care  
 RR  
 00 = X  
 01 = Y  
 10 = U  
 11 = S

Tableau I (suite et fin)

## Les instructions d'incrémentation/décrémentation, mise à zéro, complémentation.

CLR : mise à 0 d'un accumulateur ou d'une case mémoire  
 DEC : décrémentation d'un accumulateur ou d'une case mémoire

INC : incrémentation d'un accumulateur ou d'une case mémoire.

NOP : pas d'opération, incrémentation du compteur-programme.

COM : complémentation d'un accumulateur ou d'une case mémoire.

NEG : complément à 2 d'un accumulateur ou d'une case mémoire.

## Les instructions de transfert des données

LD : chargement des registres internes du CPU à partir d'une case mémoire.

ST : chargement d'une case mémoire à partir des registres internes du CPU.

PSH : empilement de (s) registre(s) sur la pile.

PUL : dépilement de (s) registre(s) depuis la pile.

EXB : échange du contenu de deux registres.

TFR : transfert de registre à registre.

## Intructions de test et de branchement

(L) BCC ou (L) BHS : branchement si pas de retenue

(L) BCS ou (L) BLO : branchement si retenue

(L) BEQ : branchement si égale à zéro

(L) BNE : branchement si différent de zéro

(L) BGE : branchement si supérieur ou égal (signé)

(L) BLT : branchement si inférieur (signé)

(L) BGT : branchement si supérieur (signé)

(L) BLE : branchement si inférieur ou égal (signé)

(L) BHI : branchement si supérieur (non signé)

(L) BLS : branchement si inférieur ou égal (non signé)

(L) BHI : branchement si négatif

(L) BPL : branchement si positif

(L) BVC : branchement si pas de débordement

(L) BVS : branchement si débordement

## Instruction de saut et de branchement

(L) BRA : branchement inconditionnel

(L) BRN : non branchement

(L) BSR : branchement à sous-programme

JMP : saut inconditionnel à une adresse effective

JSR : saut à un sous-programme

RST : retour de sous-programme

## Instruction opérant sur les pointeurs

LEA : chargement d'un registre avec une adresse effective

## Traitement des interruptions

CWAI : validation puis attente d'une interruption

SYNC : synchronisation du logiciel avec une ligne d'interruption

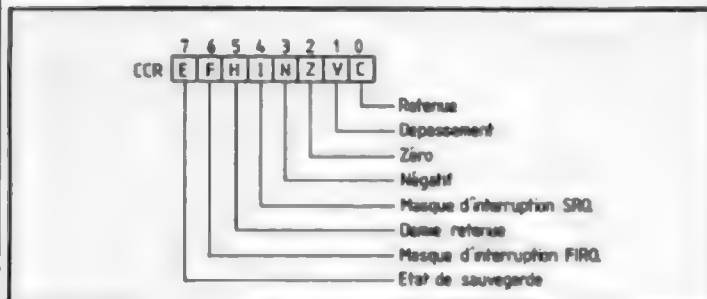
RTI : retour de sous-programme d'interruption

SWI1/SWI2/SWI3 : interruptions logicielles.

## LES FLAGS

Les flags (drapeau en français) sont les bits du registre d'état (CCR) du Microprocesseur.

La fonction de chacun des bits de ce registre est la suivante :



### Indicateurs arithmétiques

Ce sont les bits 0, 1, 2, 3 et 5 du registre d'état (CCR). Ils sont positionnés en fonction du résultat des instructions qui manipulent des données.

- Rôle du bit C (Carry = retenue)

Ce bit est positionné à 1 ou à 0 lors d'une opération arithmétique.

- Rôle du bit V (débordement en Complément à deux)

Ce bit est positionné si le résultat d'une opération arithmétique en complément à deux déborde.

- Rôle du bit Z (zéro)

Ce bit est positionné lorsque le résultat de l'opération précédente est nul.

Attention, cet indicateur est également positionné par les opérations de chargement (ex. LDA) et de stockage (ex. STA).

- Rôle du bit N (négatif)

Ce bit est positionné à la valeur du bit de poids fort du résultat d'une opération. En effet, un nombre en complément à deux est négatif si le bit de poids le plus fort est à 1.

- Rôle du bit H (Half carry → demi-retenue)

Ce bit est positionné à 1 si lors d'une opération une retenue passe du bit 3 au bit 4.

Ce bit est l'indicateur de retenue du bit 3.

### Les indicateurs d'interruption

Les indicateurs 4, 6 et 7 sont liés au fonctionnement en interruption.

- Rôle du bit I (Interrupt Mask → masque d'interruption)

Ce bit est positionné par l'utilisateur à 1 et interdit le traitement des interruptions IRQ.

- Rôle du bit F (Fast Interrupt Mask → masque d'interruption rapide)

Ce bit a le même rôle que le précédent mais pour les interruptions FIRQ.

- Rôle du bit E (sauvegarde des registres dans la pile)

Ce bit est l'indicateur de sauvegarde des registres dans la pile.

E = 1 → tout le contexte est sauvegardé

E = 0 → une partie seulement des registres est sauvegardée.

## LES MODES D'ADRESSAGE

### Mode d'adressage inhérent

Ce mode d'adressage est appelé «inhérent» du fait que le microprocesseur sait automatiquement quelle est l'opération à effectuer ainsi que les registres qui sont concernés.

Exemple :

Instruction	Opération effectuée	Registres
ABX code 3A	addition	$B + X \rightarrow X$
ASLA code 48	Décalage à gauche	A
ASLB code 58		B
ASRA code 47	Décalage à droite	A
ASRB code 57		B
CLRA code 4F	Remise à zéro	A
CLRB code 5F		B
COMA code 43	Complémentation	A
COMB code 53		B
Etc...		

Pour illustrer ce mode d'adressage, je vous propose de réaliser ce qui suit :

- RST →

- REG b → entrer 01 modifier le contenu du registre b

- Reg X → entrer 0009 modifier le contenu du registre X.

Puis entrer le programme suivant :

0100 ABX 3A Addition de B + X

0101 SWI 3F Retour au moniteur.

Lancez maintenant le programme par RST, GO 0100. Vous pouvez maintenant vérifier le contenu du registre X. Je vous parie qu'il y a 0009.

L'instruction ABX est effectivement l'addition du registre B avec X et le résultat est mis dans X.

### Comment travaille l'unité centrale avec les mémoires ?

Pour que l'unité centrale «s'y retrouve», il faut lui indiquer dans les instructions codées :

- soit où aller chercher les données à traiter ;

- soit où aller stocker les résultats du traitement.

Dans ce but, le circuit 6809 possède dix modes d'adressage, c'est-à-dire dix façons de coder les adresses, ce qui fait de lui le plus puissant des microprocesseurs 8 bits. Il n'est pas question d'étudier dans le détail chacun des dix modes d'adressage. Nous ne présenterons que certains d'entre eux. Les ouvrages cités en bibliographie à la fin permettront de compléter cette introduction.

### Programme n°2 : Adressage étendu

La façon la plus simple de donner une adresse est de l'écrire «en clair», c'est-à-dire à l'aide d'un nombre de quatre chiffres hexadécimaux : c'est le mode d'adressage «étendu».

Exemple : transférer une donnée de la case-mémoire n°0210 à la case-mémoire n°0230. En se rappelant que ce transfert ne peut avoir lieu directement mais que la donnée

doit transiter par l'unité centrale, on peut illustrer ce processus de la façon suivante (fig. 15).

L'ordinogramme et le programme correspondants sont simples à écrire :

Un autre mode d'adressage consiste à n'écrire dans le programme qu'une partie du numéro de l'adresse (les deux chiffres hexadécimaux de puissance inférieure, à droite), l'autre partie (les deux chiffres hexadécimaux de puissance

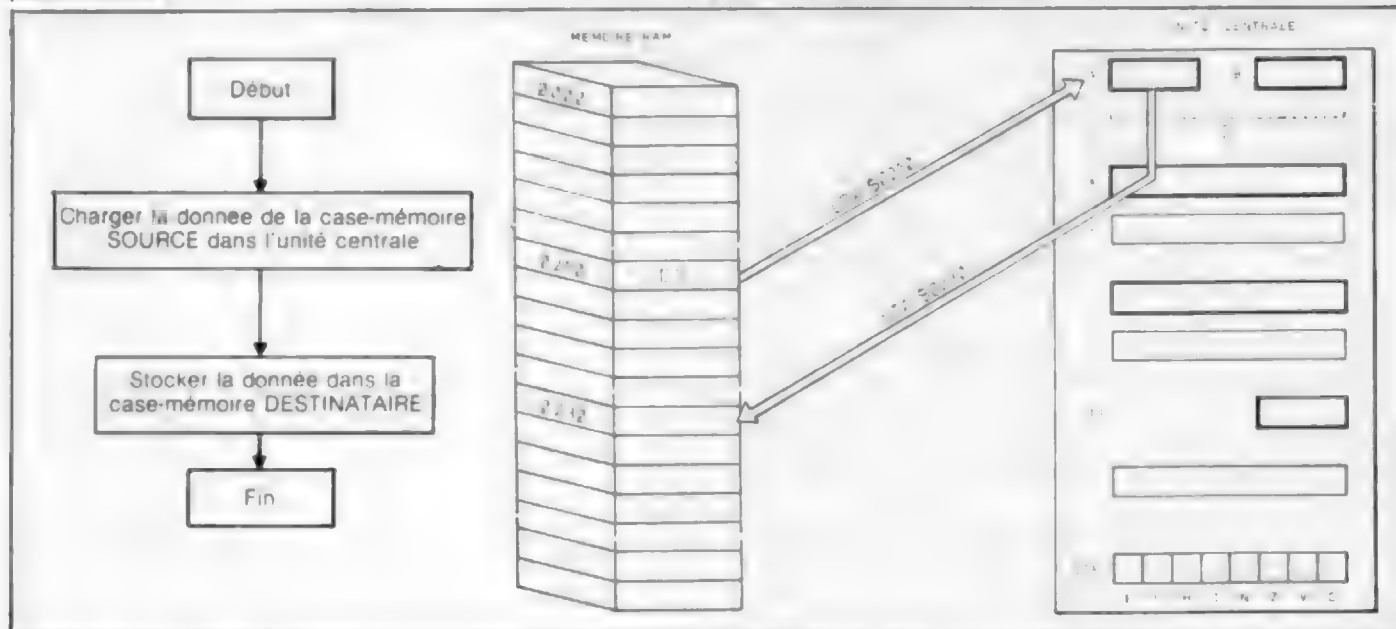


Fig. 15 : Transition des données.

Adresse-programme	Langage machine code-instruction	Langage Assembleur symbolique opération opérandes			Commentaires
0 0 2 0	B 6 0 2 1 0		LDA	\$0210	Amène la donnée dans l'unité centrale
0 0 2 3	B 7 0 2 3 0		STA	\$0230	Stockage dans nouvel emplacement-mémoire
0 0 2 6	3 F		SWI		Arrêt - Retour au moniteur (visu registres)
					Fin

Il ne vous reste plus qu'à entrer le programme en code-machine, à le lancer et à vérifier le transfert de la donnée en comparant le contenu des cases-mémoires 0210 et 0230 avant et après exécution du programme.

#### Remarques

1. Pour trouver le code correspondant aux instructions de chargement et stockage en mode d'adressage étendu, il suffit de consulter le tableau 1.
2. Avec le Microkit 09, seul le langage machine (code hexadécimal) est utilisable.
3. En langage assembleur, une valeur écrite en hexadécimal est précédée du symbole \$ (dollar).

#### Programme 3, 4 et 5 : Adressage direct

supérieure, à gauche) étant contenus dans un registre dit de «page directe» (DPR «Direct Page Register»). Il suffira alors à la machine de rassembler les deux parties pour reconstituer l'adresse complète. Dans ce mode d'adressage, les 64K, soit 65 536 numéros de cases-mémoires sont ainsi repérés par 256 pages de 256 numéros chacune.

#### Programme 3 :

Additionner deux nombres hexadécimaux entrés au clavier, à l'aide du programme-moniteur dans les cases-mémoires n° 0240 et 0241. Stocker le résultat en 0242.

Pour cela, il nous faut charger les deux chiffres (le nombre 02 de la partie supérieure de l'adresse) 0 puis 2 dans le «registre de la page» en utilisant le petit programme suivant. Consulter le tableau 1 pour retrouver les codes utilisés.

Adresse-programme	Langage machine code-instruction	Langage Assembleur symbolique opération opérandes			Commentaires
0 0 3 0	C 6 0 2		LDB	#02	Charger «immédiatement» la valeur 02 dans le registre B
0 0 3 2	1 F 9 B		TFR	B,DP	La transférer pour initialiser le registre de page

0 0 3 4	9 6 4 0		LDA	< \$40	Chargement du 1 <sup>er</sup> nombre dans unité centrale
3 6	9 B 4 1		ADDA	< \$41	Addition avec le 2 <sup>e</sup> nombre
3 8	9 7 4 2		STA	< \$42	Stockage du résultat
3 A	3 F		SWI		

Il faut noter qu'il n'existe pas d'instruction de chargement «immédiat» dans le registre DP ; dans la suite du programme, il suffira de charger le nombre correspondant à la partie inférieure de l'adresse dans l'unité centrale et d'utiliser une instruction d'addition en mode d'adressage direct, la machine se chargeant de reconstituer l'adresse. Les codes utilisés se retrouvent dans le tableau 1 dans la colonne «adressage direct».

Noter que l'adressage direct est symbolisé par < en langage Assembleur. La figure 16 représente le séquençement des opérations effectuées. Chaque fois, l'adresse complète «source» ou «destination» s'obtient en rassemblant les chiffres du registre DP et les chiffres contenus dans le programme.

L'intérêt de ce mode d'adressage est de n'utiliser que deux chiffres hexadécimaux d'adresse (soit 1 octet de huit chiffres binaires, chaque «quartet» de quatre chiffres binaires codant un chiffre hexadécimal), ce qui économise de la place pour des opérations concernant une zone-mémoire, dont les numéros d'adresse sont situés dans la même «page».

#### Programme 4

Additionner deux nombres décimaux entrés par clavier, à l'aide du programme-moniteur. Ce programme implique :

- de rentrer des nombres comportant seulement des chiffres de 0 à 9, c'est-à-dire des nombres décimaux ;
- d'effectuer dans le programme après l'addition, un ajustement décimal, à l'aide de l'instruction DAA, de façon à obtenir le résultat en décimal. A noter que DAA ne fonctionne que derrière une instruction d'addition et n'agit que sur l'accumulateur A.

Nous vous proposons d'essayer d'écrire ce programme seul, avant de le comparer au listing des programmes présentés.

#### Programme 5

Additionner deux nombres de 16 bits (2 octets). Il faut alors utiliser deux cases-mémoires pour écrire chaque nombre, et l'accumulateur D (composé par l'assemblage des accumulateurs A et B) pour effectuer l'opération.

#### Programmes 6, 7 : Adressage immédiat

L'adressage «immédiat» (que nous avons utilisé sans l'expliquer à la première ligne du programme 3) consiste à intro-

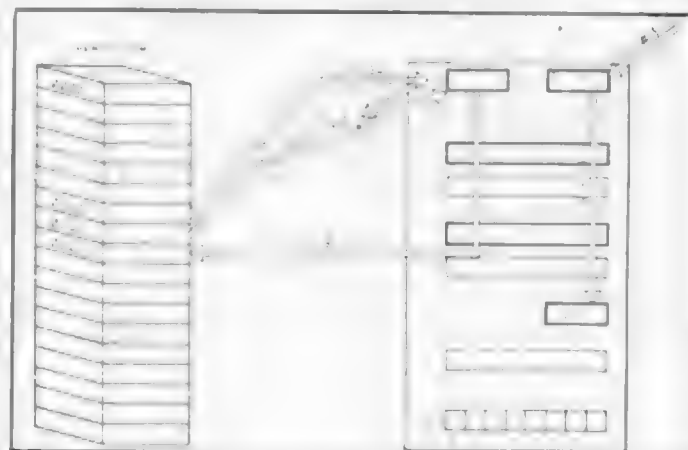


Fig. 16.

duire directement une valeur, indiquée dans le programme, dans un registre spécifié par le code-instruction.

#### Programme 6

Additionner deux nombres hexadécimaux entrés par programme dans les cases-mémoires 0240 et 0241.

Nous pouvons stocker les nombres en utilisant l'adressage immédiat (symbolisé par # en langage Assembleur) et l'adressage direct (symbolisé par < en langage Assembleur) suivant le programme ci-dessous et la figure 17.

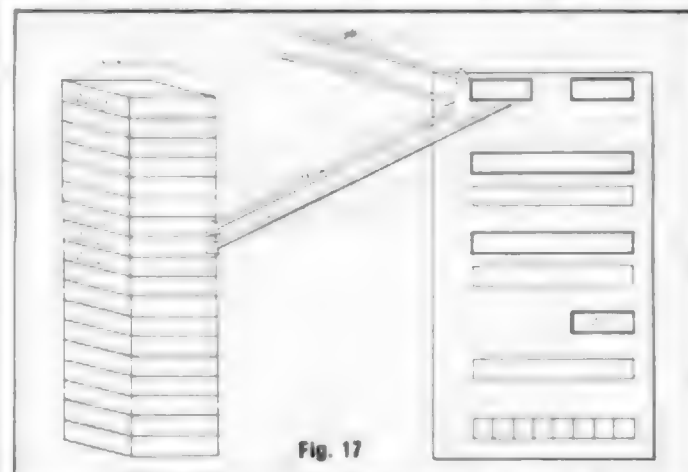


Fig. 17

Adresse-programme	Langage machine code-instruction	Langage Assembleur symbolique opération opérandes			Commentaires
0 0 6 0	8 6 0 3		LDA	#\$03	} Stockage du 2° nombre
6 2	9 7 4 0		STA	< \$40	
6 4	8 6 0 4		LDA	#\$04	} Stockage du 1° nombre
6 6	9 7 4 1		STA	< \$41	

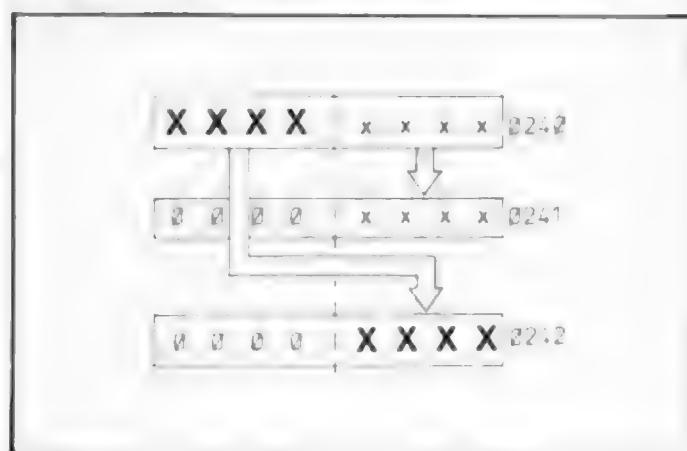


La suite du programme d'addition est similaire à la deuxième partie du programme 3.

0   0   6   8	9   6   4   0		LDA	< \$40	
6   A	9   B   4   1		ADDA	< \$41	
6   C	9   7   4   2		STA	< \$42	
6   E	3   F		SWI		

### Programme 7

Séparer l'octet (huit chiffres binaires) inscrit en deux quartets (quatre chiffres binaires) à ranger en 0241 et 0242 suivant le schéma suivant :

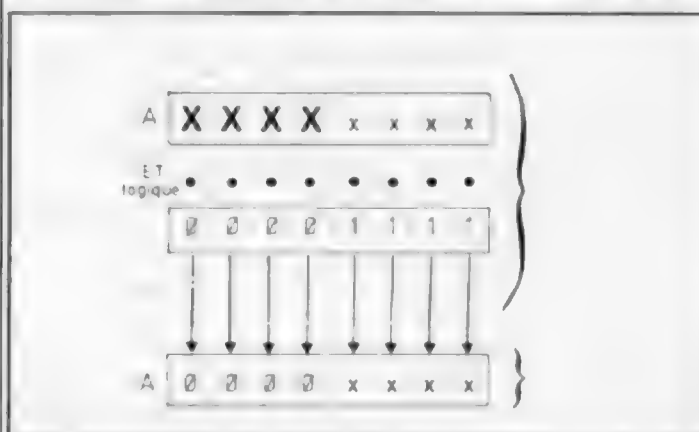


Pour cela, après avoir chargé le nombre dans le registre A (ou B), nous allons utiliser des instructions d'opération logique avec adressage immédiat, et de décalage de chiffres binaires. Mais écrivons d'abord l'ordinogramme puis le programme :



Adresse-Programme	Langage machine code-instruction	Langage Assembleur symbolique	
		opération	opérandes
0   0   7   0	9   6   4   0	LDA	< \$40
1   7   2	8   4   0   F	ANDA	#%00001111
2   7   4	9   7   4   1	STA	< \$41
3   7   6	9   6   4   0	LDA	< \$40
4   7   8	4   4	LSRA	
5   7   9	4   4	LSRA	
6   7   A	4   4	LSRA	
7   7   B	4   4	LSRA	
8   7   C	9   7   4   2	STA	< \$42
9   7   E	3   F	SWI	

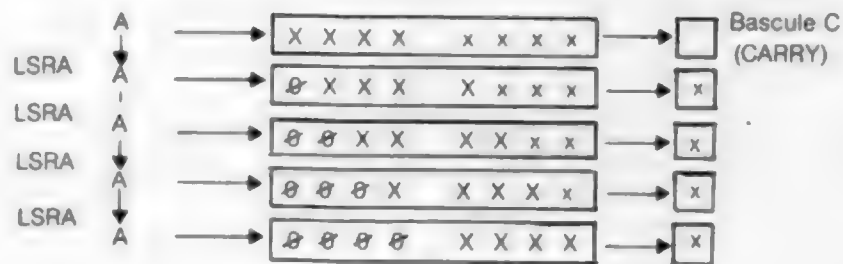
L'opération de masquage se réalise de la façon suivante :



On effectue un ET logique entre chaque chiffre binaire de l'octet mis dans A et le chiffre binaire correspondant de la valeur introduite en adressage immédiat avec l'instruction. ANDA # % 00001111. Le symbole % indique que la valeur est écrite en binaire dans le programme en langage Assembleur. On pourrait l'écrire en hexadécimal (\$0F).

On a donc «masqué» le quartet de droite en le remplaçant par des 0.

L'instruction du décalage LSRA permet de décaler le quartet de gauche sur la droite et de le remplacer par 0.



### Programme 8 et 9 : Adressage relatif ou la «Puce sauteuse»

Pour expliciter ce mode d'adressage, supposez que vous demandiez le numéro de la chambre d'un ami à un garçon d'hôtel. Le garçon pourra vous l'indiquer en disant :

- «c'est la chambre n°1221» (adressage «étendu») avec 12 indiquant l'étage et 21 le numéro de la chambre ;
- ou «c'est la chambre n°21» (adressage «direct») à supposer qu'il vous ait indiqué auparavant l'étage, ou bien que vous vous trouviez déjà au 12<sup>e</sup> ;
- ou «c'est la quatrième porte avant (ou après) celle-ci», c'est l'adressage «relatif» ;

- ce mode d'adressage est utilisé dans les programmes lorsqu'on veut sauter (pour une puce programmée **c'est naturel !**) d'un endroit du programme à l'autre, ou autrement dit «se brancher» à une adresse différente. On indiquera alors non pas en absolu l'adresse de destination mais la distance qui vous en sépare. Les deux exemples qui suivent illustrent ce mode.

#### Programme 8

Trouver le plus grand des deux nombres stockés en 0240 et 0241 et le ranger en 0242.

On va évidemment utiliser ici une instruction de comparaison, mais suivant le résultat de la comparaison, il faudra ranger soit l'un, soit l'autre des deux nombres. C'est ce qu'illustre l'ordinogramme ci-dessous :

On voit donc que si l'on répond positivement à la question «le 1<sup>er</sup> nombre est-il plus grand que le 2<sup>e</sup> ?», il faut alors se brancher plus loin dans le programme et sauter la partie du programme exécutée dans le cas où l'on répondrait négati-

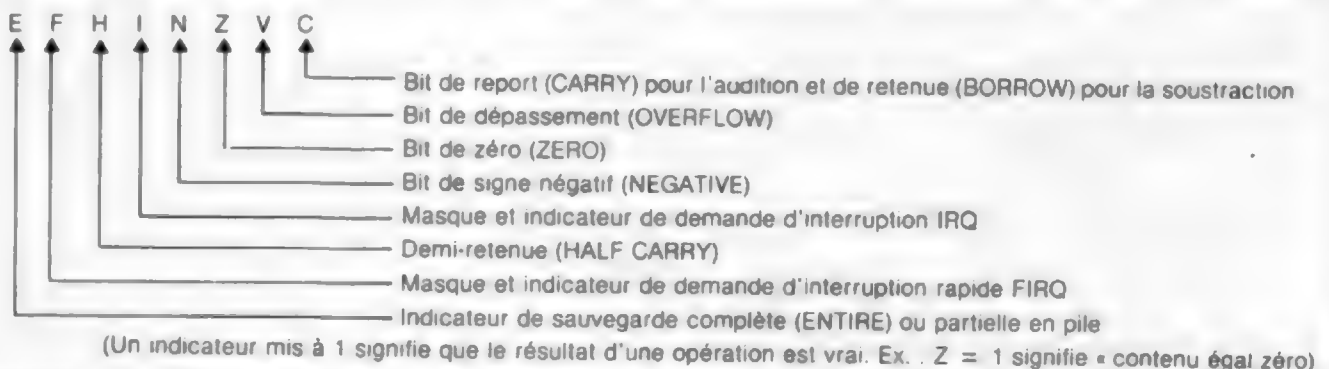
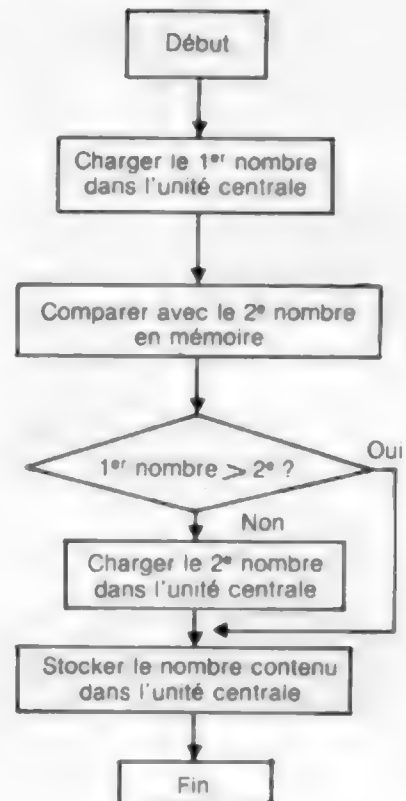


Fig. 18 : Registre d'état CCR.

vement. Mais qui nous indique le résultat positif et négatif de la comparaison ? De combien de pas devons-nous sauter si c'est positif ?

Il existe dans l'unité centrale des bascules-indicateurs, nous donnant des renseignements sur le résultat d'opérations. Ces bascules sont rassemblées dans un registre CCR («Code Condition Register»), appelé aussi «registre d'état et des indicateurs d'états» et décrit en figure 18.

Pratiquement, lorsqu'on écrit une instruction demandant un branchement dans le programme en fonction du résultat d'une opération, la machine va tester les indicateurs C, V, Z et N.

Les indicateurs I, F et E sont utilisés dans les programmes d'interruption.

Ecrivons maintenant le programme de comparaison :

Adresse-programme	Langage machine code-instruction	Langage Assembleur symbolique	Commentaires
0 0 18 0	9 6 14 0	LDA <\$40	Charger le 1 <sup>er</sup> nombre dans l'unité centrale
0 0 18 2	9 1 14 1	CMPS <\$41	Le comparer au 2 <sup>e</sup>
0 0 18 4	2 4 0 0	BHS 00	Si 1 <sup>er</sup> nombre plus grand se brancher
0 0 18 6	9 6 14 1	LDA <\$41	Charger le 2 <sup>e</sup> nombre
0 0 18 8	9 7 14 2	00 STA <\$42	Stocker le nombre le plus grand
0 0 18 A	3 F	SWI	Fin

On a volontairement laissé en blanc la valeur du saut à effectuer dans le programme, valeur à écrire après l'instruction BHS qui ordonne le branchement dans le cas où le résultat de la comparaison (qui équivaut à une soustraction) est positif ou nul.

#### Comment calculer le déplacement en adressage relatif ?

1. Lorsque la machine a décodé l'instruction BHS suivie de la valeur du déplacement que nous allons calculer, le compteur-programme qui indique où on est dans le programme pointe alors l'adresse de l'instruction suivante : soit 0086. Il faut donc faire sauter le compteur programme de deux pas vers l'avant. La valeur du déplacement à inscrire ici à l'adresse 0085 est donc de 0 2.

2. On peut imaginer un compteur situé dans les emplacements laissés en clair 00 à l'adresse 0085, et initialisé à FF. Pour atteindre l'adresse de destination, ici 0088, on va s'y transférer en incrémentant à chaque passage d'adresse le compteur de 1.

Ce qui donne l'évolution suivante : 0085 — 0086 — 0087 — 0088. Compteur : FF — 00 — 01 — 02.

3. Solution «relaxe» : il existe dans le programme-moniteur de la maquette Micro Kit 09 un «calcul automatique du

déplacement» pour ce mode d'adressage.

Ce calcul s'effectue à l'aide de la touche ofs «offset».

#### Programme 9

Décompter de 255 à 000.

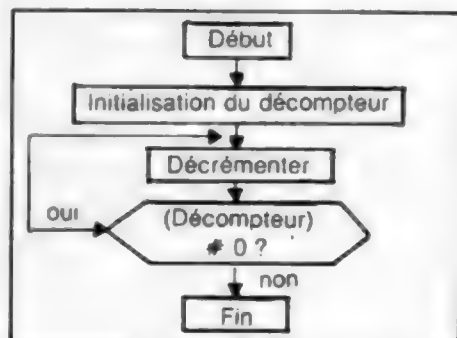
L'ordinogramme et le programme sont très simples et reproduits ci-dessous. Il suffit de créer une «boucle» logique tant que le contenu du registre-décompteur n'est pas nul.

— Premier calcul du déplacement :

Après décodage de l'instruction demandant un branchement à l'instruction décomptage tant que le décompteur n'est pas nul, le compteur d'adresses-programme (PC) se trouve en 0095, il devra sauter en 0092 soit un saut en arrière de -3. Mais comment écrire -3 en hexadécimal ? En utilisant le code dit «complément à 2». Le signe «-» sera codé par le chiffre binaire 1, placé le plus à gauche puis au lieu d'écrire la valeur 3 on écrira son complément à 2<sup>7</sup>, soit  $128 - 3 =$  ce qui s'écrit en binaire :

1 111 1101  
— signe complément à 2<sup>7</sup> de 3

soit FD en hexadécimal.



Adresse	Langage machine code-instruction	Langage opération	Assembleur symbolique opérandes
0 0 9 0	8 6 F F	LDA	\$FF
0 0 9 2	4 A	DECA	
0 0 9 3	2 6 0 0	BNE	00
0 0 9 5	3 F	SWI	

- Deuxième calcul de déplacement :

En reprenant la méthode du compteur placé en 0094, et initialisé à FF puis déplacé et décrémenté progressivement jusqu'en 0092 on retrouve facilement la valeur F D.

- Troisième calcul :  
avec III touche offs.

### Programmes 10 à 12 : Adressage indexé

Un autre mode d'adressage, très varié, consiste à calculer l'adresse effective de la case-mémoire, où on va soit chercher soit stocker une donnée, à partir d'une adresse de base à laquelle s'ajoute un déplacement.

Ainsi, si l'adresse de base est 0200, pour aller en 0240, il suffira d'indiquer un déplacement de \$40. L'adresse de base sera inscrite dans un registre X, Y ou même U ou S ou éventuellement D.

### Programme 10 : Recopie de zone-mémoire

L'on désire recopier les données inscrites entre les adresses 0200 à 0207 dans une zone-mémoire située entre 0208 et 020F. Le principe de cette recopie est schématisé ci-dessous :

On note que :

. la donnée transite par le registre A ;

. l'adresse de base (qui est l'adresse de début de la zone-mémoire source) est inscrite dans le registre X appelé aussi

registre-index, car «pointant» une adresse ;

. l'adresse effective du destinataire, qui correspond au début à la première adresse de la zone-mémoire destinataire est donc égale à la source + 8.

Nous donnons ci-contre l'ordinogramme puis le programme en langage assembleur symbolique. Amusez-vous à l'écrire en langage machine en vous aidant des remarques qui suivent, avant de le comparer à celui proposé.

### Remarques (1)

Ce mode d'adressage est **indexé à déplacement nul**. L'adresse effective est égale à : (l'adresse base/inscrite dans X) + (0). Cette instruction se code en s'aidant du tableau 1 et du tableau 2 (reproduit ci-dessous).

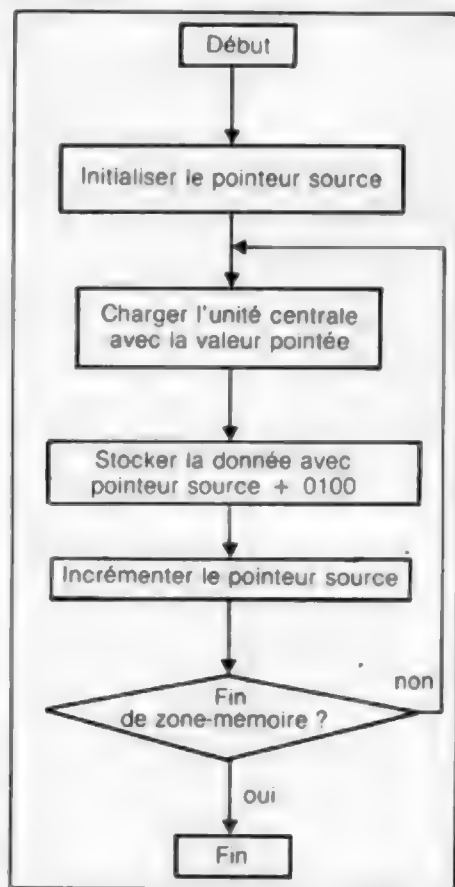
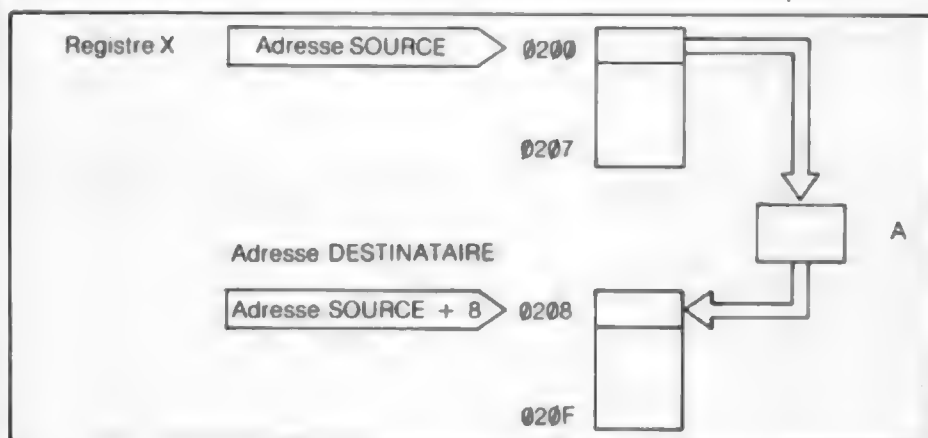
LDA : en adressage indexé se code A6 ;

X : signifie que l'adresse de base est inscrite dans X et qu'on lui ajoute un déplacement nul. La première ligne du tableau 2 (mode non indirect) nous indique le code 1RR00100 où RR est le code du registre choisi, soit 00 pour X. Le code 84 en hexadécimal ainsi obtenu est appelé le «post-octet» (car venant après le ou les octet(s) du code opération de l'instruction), et spécifie le mode d'adressage indexé.

(2) : L'adresse effective de stockage est ici obtenue par un adressage **indexé à déplacement constant** égal à 8 (écrit ici en décimal, non précédé du symbole \$).

STA : se code en adressage indexé A7 ;

Adresse-programme	Langage machine code-instruction		Langage assembleur symbolique Opération	Opérandes	Remarques
0 1 0 0			LDX	#\$0200	
			LDA	,X	(1)
			STA	8,X	(2)
			LEAX	1,X	(3)
			CMPX	#\$0208	
			BNE	□ □	(4)



Type	Formes	Mode non indirect				Mode indirect			
		Syntaxe assembleur	Post-octet code OP	$\sim$	N	Syntaxe assembleur	Post-octet code OP	$\sim$	N
Déplacement constant à partir de R (signé)	pas de déplacement déplacement 5 bits déplacement 8 bits déplacement 16 bits	R d, R d, R d, R	1RR00100 0RRnnnn 1RR01000 1RR01001	0 0 1 0 1 1 4 2		[, R] par défaut — 8 bits [d, R] [d, R]	1RR10100 1RR11000 1RR11001	3 0 4 1 7 2	
Accumulateur utilisé comme déplacement pour le Registre R (déplac. signé)	registre de déplac. A registre de déplac. B registre de déplac. D	A, R B, R D, R	1RR00110 1RR00101 1RR01011	1 0 1 0 4 0		[A, R] [B, R] [D, R]	1RR10110 1RR10101 1RR11011	4 0 4 0 7 0	
Auto incrémentation/décrémentation du registre R	incrémenté par 1 incrémenté par 2 décrémenté par 1 décrémenté par 2	R + R + + R R	1RR00000 1RR00001 1RR00010 1RR00011	2 0 3 0 2 0 3 0		impossible [R + +] impossible [R]	1RR10001 1RR10011	6 0 6 0	
Déplacement constant à partir de PC	déplacement 8 bits déplacement 16 bits	d, PCR d, PCR	1XX01100 1XX01101	1 1 5 2		[d, PCR] [d, PCR]	1XX11100 1XX11101	4 1 8 2	
Indirect étendu	adresses 16 bits	—	—	—		[d]	10011111	5 2	

#### Légende :

R = registres X, Y, U ou S

X = valeur indifférente

$\sim$  = nombre de cycles-horloge additionnels

N = nombres d'octets additionnels

d = valeur du déplacement (décimal)

RR = code registre 00 = X

01 = Y

10 = U

11 = S

nnnn = valeur du déplacement en complément à 2

(d'après documents Motorola/EFCIS)

**Tableau 2 : Types d'adressage indexé et codage du post-octet.**

8,X : se code (voir deuxième ligne du tableau 2).

$$\begin{array}{c} \text{ORR } n \text{ nnnn} \\ \text{Code } \text{Valeur du déplacement} \\ \text{registre codé en complément à 2} \end{array}$$

On rappelle que dans le code complément à 2, le chiffre binaire le plus à gauche indique le signe (0 pour +, 1 pour -), et qu'on écrit une valeur positive en code binaire pur et une valeur négative avec son complément à 2<sup>n</sup>.

Ici, le déplacement de valeur 8 étant compris entre - 16 et + 15, quatre bits suffisent pour coder sa valeur, plus un bit de signe.

Si la valeur du déplacement est comprise entre - 128 et + 127 il faudra ajouter à l'octet (ou aux deux octets) codant l'opération, et au post-octet (voir troisième ligne du tableau) un octet supplémentaire codant la valeur du déplacement. Voir la colonne N du tableau 2.

Deux octets supplémentaires seront nécessaires pour une valeur de déplacement comprise entre - 32 768 et + 32 767.

Sur la maquette Microkit 09, le calcul du code hexadécimal des post-octets contenant la valeur du déplacement est automatisé.

Exemple : nous voulons coder l'instruction LDA - 17,X (- 17 étant exprimé en décimal).

En cours d'écriture du programme du clavier :

1. Entrer le code A 6 pour «LDA»

2. Lecture du code du post-octet indiquant que l'on utilise le registre d'index X pour un déplacement codé sur 8 bits : en effet, pour coder - 17 en code complément à 2, il faut au moins cinq chiffres binaires significatifs et un bit de signe. On choisit donc la troisième ligne du tableau, correspondant à un déplacement codé avec 8 bits :


1RR0 1000 soit avec le registre d'index X, 1000 1000 = \$88.

- Entrer 8 8.


3. Calcul automatique du déplacement :

- appuyer sur X puis sur Ofs l'affichage présente alors :

0000 88

- Préciser le sens du déplacement en appuyant : soit sur INC pour un déplacement positif — affichage de P soit sur Dec pour un déplacement négatif — affichage de  ... dans cet exemple, on appuie donc sur Dec.
- Entrer la valeur du déplacement 00017.
- Appuyer sur GO.

Le programme-moniteur calcule alors le code du déplacement, soit ici EF, et le place dans la (les) case(s) mémoire suivante(s) de notre programme. L'instruction LDA-17, X est donc codée avec : A6 88EF. Pas de panique, si vous vous trompez dans la procédure de codage, la maquette devrait afficher :

  
(comme Erreur)

3. Pour incrémenter le contenu du registre pointeur X, c'est-à-dire pour incrémenter l'adresse de base, on utilise l'instruction de calcul d'une adresse effective.

LEA, associée à un adressage indexé à déplacement constant qui s'écrit en langage symbolique LEAX 1,X et qui signifie :

LEAX : «Charge dans X l'adresse effective...»

1,X : «... obtenue en prenant le contenu de X et en lui ajoutant 1».

Le code de LEAX est 30. Le code de 1,X se lit à la deuxième ligne du tableau (mode non direct).

4. La valeur du déplacement pour le branchement s'obtient suivant les indications données pour les programmes 8 et 9.

La figure ci-contre résume les types utilisés d'adressage indexé.

Question «blanche» : Faire un programme qui recopie de 01F8 à 01FF les contenus de la zone 0200 à 0207.

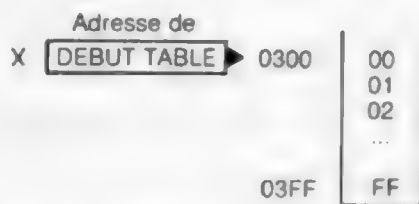
#### Programme 11

Inscrire des objets par ordre croissant dans une table-mémoire de 0300 à 03FF.

Il s'agit d'inscrire par programme :

\$ 00 dans la case-mémoire 0300





\$ 01 dans la case-mémoire 0301

\$ FF dans la case-mémoire 03FF

Adresse-programme	Langage machine code - instruction	Langage assembleur symbolique Opération	Opérandes
0, 1   1, 0		LDX	\$0300
		CLRA	
		CLRB	
		STB	D,X (1)
		INCB	
		BNE	
		SWI	

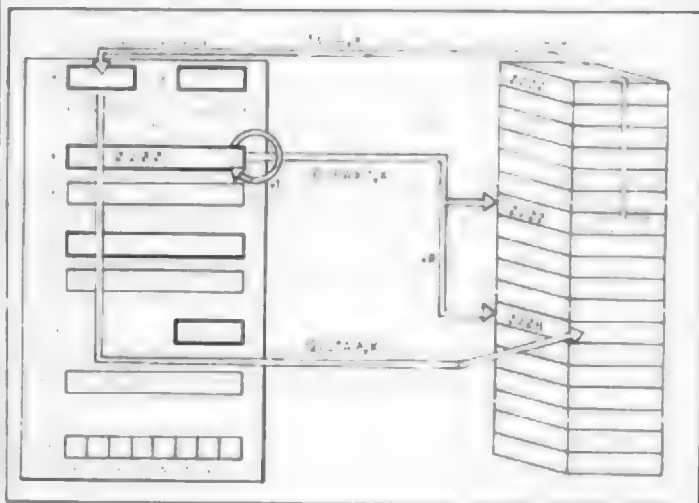
#### Remarque (1)

Dans ce mode d'adressage indexé à déplacement accumulateur la valeur du déplacement se trouve dans l'accumulateur D. En effet, le déplacement est ici lié à la valeur contenue dans le générateur d'octets.

#### Question «rouge»

Pourquoi ne pas avoir choisi simplement l'accumulateur A ou B au lieu de D comme valeur du déplacement dans l'instruction STB, D,X ?

Essayez alors votre programme modifié, avec une table-mémoire située entre 0500 et 05FF puis vérifiez le contenu de cette zone et de la zone 0480 à 04FF. Que s'est-il passé ?



#### Programme 12 :

Faire un décompteur de temps utilisant les registres X et A, initialisés respectivement à FFFF et FF. Calculer la durée totale de décomptage.

Indications :

1. L'utilisation de deux décompteurs se fait à l'aide de boucles logicielles encastrées, selon l'ordinogramme ci-contre
2. Le calcul du temps mis pour le décomptage se fait en utilisant le tableau 1 où est indiqué dans la colonne ~ le nombre de cycles horloge que dure une instruction, sachant que le cycle-horloge du Microkit 09 est de 1  $\mu$ s.

Ex : Charger le registre X avec la valeur FFFF en adressage immédiat dure 3  $\mu$ s tandis que l'exécution de SWI dure 19  $\mu$ s.

EX. : Si la boucle 2 est parcourue FF fois, le temps d'exécution de cette boucle sera 255 (2 + 3)  $\mu$ s, 2  $\mu$ s correspondant à l'instruction DEC, 3  $\mu$ s correspondant à l'instruction de branchement BNE.

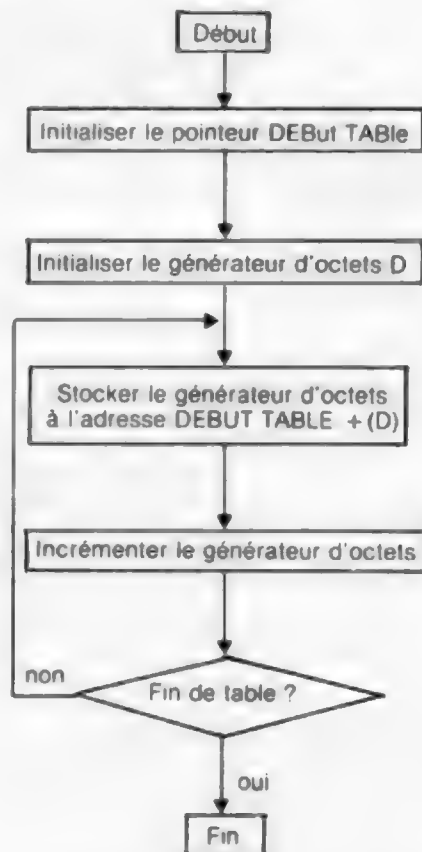
3. Si on utilise comme décompteur le registre X, il n'existe pas d'instruction «DEX» ! Mais, par contre, il existe une instruction LEAX (Load Effective Adress) qui permet de charger une valeur dans X. Cette valeur peut être l'ancienne valeur de X diminuée de 1.

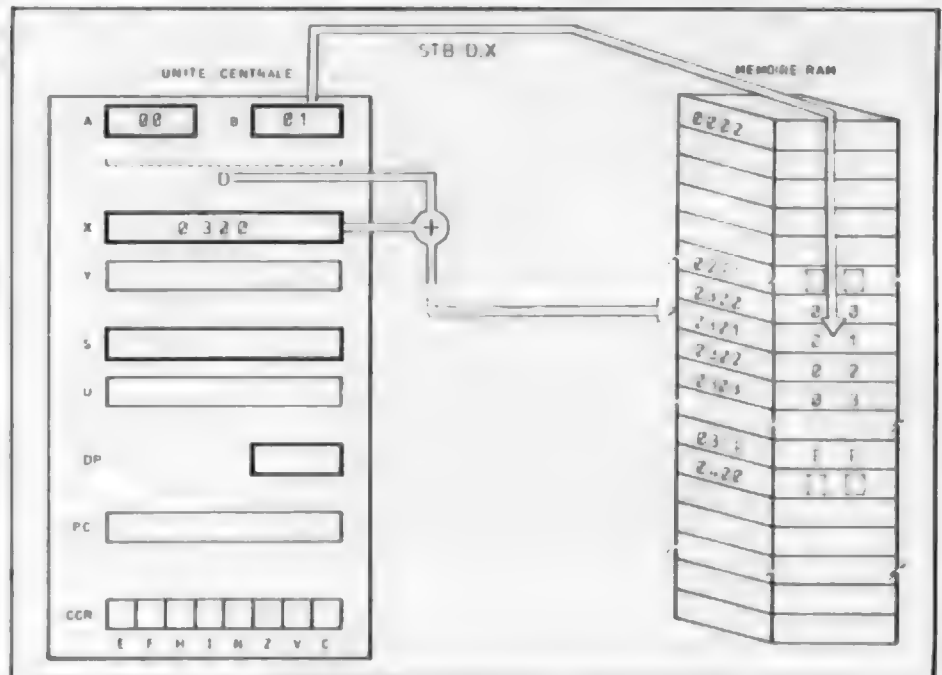
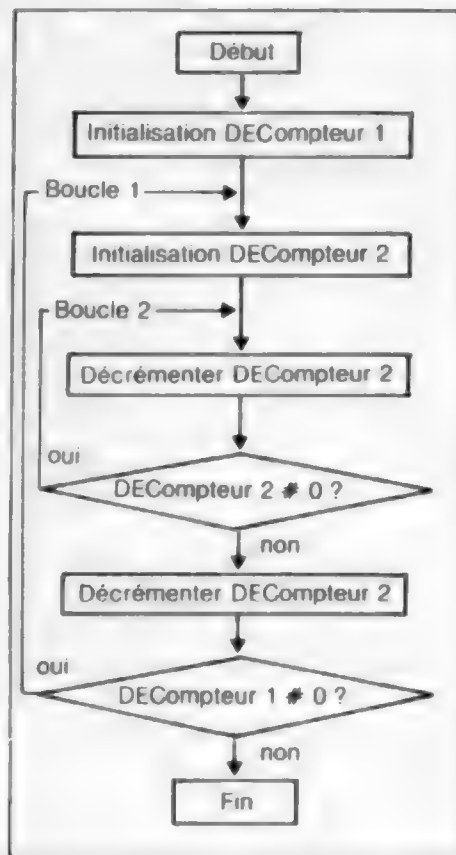
Cette instruction s'écrit en langage symbolique :

LEAX , -X

et se code :

30 , 82





Adresse programme	Langage machine code instruction			Langage assembleur symbolique	
	1 <sup>er</sup> octet	2 <sup>e</sup> octet	3 <sup>e</sup> octet	Opération	Opérandes
0 0 0 0	C	C	7 D 7 F	LDD	#7D7F
0 0 3	F	D	0 7 F A	STD	<07FA
0 0 6	C	C	7 E 3 F	LDD	#7E3F
0 0 9	F	D	0 7 F C	STD	<07FC
0 0 C	C	C	E 3 6 B	LDD	#E36B
0 0 F	F	D	0 7 F E	STD	<07FE
1 1 2	B	D	E 0 7 B	JSR	\$E07B
1 1 5	2	0	F B	BRA	
1 1 7					

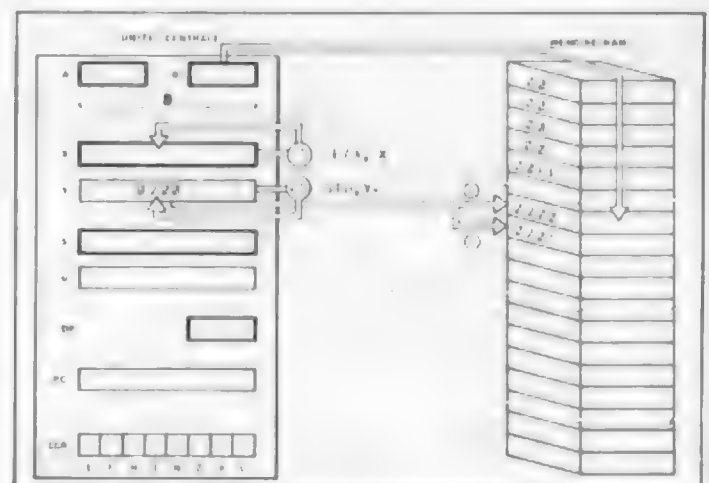


Fig. 19 : Auto-incrémentation.

Son temps d'exécution est de :

$(4 + 2) \mu s$ .

Elle signifie

LEAX : «Charge dans X l'adresse effective...»

, - X : «... obtenue 1<sup>o</sup> en prédécrémentant le contenu de X, puis 2<sup>o</sup> en rechargeant cette nouvelle valeur».

Cette instruction utilise le mode d'**adressage indexé à autopredécrémentation**. Il existe aussi des possibilités de double prédécrémentation, (-- R) et de post-incrémentation automatiques, R + et ,R+ +. Se reporter au tableau 2 pour le codage de ce type d'adressage schématisé en figure 19.

4. Le listing complet du programme et le temps mis pour son exécution se trouvent en fin d'article... Mais essayez d'abord de le mettre au point par vous-même.

Vérifier le temps indiqué avec votre montre en chronométrant le temps écoulé entre le lancement du programme (l'affichage du Microkit 09 s'éteint) et le moment où il se rallume (après avoir rencontré l'interruption SW1).

Comme cela apparaît dans le tableau 2, il existe d'autres types d'adressage indexé, sans parler du mode indirect. Dans ce mode, l'unité centrale «va chercher chez Dupont l'adresse de Durand».

Nous allons maintenant voir comment l'unité centrale travaille non seulement avec les mémoires mais avec les périphériques, via le coupleur d'entrée/sortie (circuit 6821 en

haut à gauche de la carte centrale).

Pour cela, introduisez le programme suivant ci-contre en mémoire RAM et faites-le exécuter.

Après avoir lancé le programme, vous devez voir apparaître

« 8809 RR »

sur les afficheurs de la carte périphérique. Ce programme fait donc travailler :

- l'unité centrale 6809 qui décode et exécute les instructions ;
- la mémoire RAM (adresses 0000 à 07FF), car c'est dans les cases-mémoire 07FA à 07FF que le programme stocke les six octets de code précédemment et consécutivement chargés dans le double accumulateur D ;
- la mémoire ROM (adresses E000 à E7FF), car l'instruction JSR (Jumping to SubRoutin) appelle un « sous-programme » qui démarre à l'adresse E07B et dont l'exécution fait allumer (ou non) les segments des six afficheurs de la carte périphérique en fonction du code contenu dans les six cases-mémoires de la RAM ;
- le circuit coupleur d'entrée/sortie 6821 (« Péripheral interface adapter : PIA »), car il interface le bus des données de la carte centrale à l'affichage et au clavier de la carte périphérique ;
- les six afficheurs et le circuit de sélection.

La figure 20 schématise le fonctionnement du processus d'affichage.

Avant de l'analyser en détails, voici quelques remarques :

- (1) Les huit fils du bus de données sont reliés aux sept segments lumineux et au point décimal des afficheurs, à travers le coupleur entrée/sortie et un circuit « buffer » 74LS240. L'interconnexion est faite selon l'ordre ci-dessous.



Si, sur le bus de données se trouve le code \$7F (soit 0111 1111 en binaire), tous les segments vont être allumés, mais non le point décimal. Le symbole E sera donc affiché.

Le code \$7E (soit 0111 1110 en binaire) fera apparaître le symbole 0, le segment central n'étant pas allumé.

Il est alors facile de déduire les codes d'allumage 7 segments des symboles hexadécimaux (tableau 3).

Cette table appelée DIGTBL (« Digit table ») = table des afficheurs) est en fait mémorisée en mémoire ROM à partir de l'adresse E010 mais n'est pas utilisée dans notre programme.

On en déduit facilement le code 7 segments des symboles P et P (comme « micro-processeur »).

- (2) Pour des raisons de séquençement programmé (un seul code d'allumage présent à la fois sur le bus de données et destiné à un seul afficheur) et de persistance rétinienne (il suffit de rafraîchir le contenu d'un afficheur d'au moins 25 fois par seconde), les codes d'allumage 7 segments ne sont pas mémorisés en permanence dans chaque afficheur

mais multiplexés... d'où économie d'énergie.

Le programme d'affichage placera donc à tour de rôle un code d'allumage 7 segments sur le bus de données pour allumer un afficheur. La sélection de l'afficheur se fait à l'aide d'un code mis dans le registre B du coupleur d'entrée/sortie, code qui activera la sortie correspondante du circuit décodeur 7442.

- (3) Les six codes correspondant aux six afficheurs doivent être stockés dans une zone réservée de la mémoire RAM. Cette zone-mémoire est située entre les adresses 07FA à 07FF, chaque case-mémoire contenant le code d'allumage 7 segments d'un afficheur particulier selon la répartition ci-dessous (fig. 20).

L'ensemble de ces six cases-mémoires est appelé DISBUF (« Display buffer » = registre tampon d'affichage).

- (4) L'ordinogramme du programme d'affichage de « 6809 µP » que nous avons fait exécuter peut se représenter ainsi (fig. 21).

On peut voir dans cet ordinogramme que le programme principal après avoir stocké les codes d'allumage des symboles « 6809 µP » dans la zone DISBUF, appelle un sous-programme qui est déjà inscrit dans la mémoire ROM à partir de l'adresse E07B. Ce sous-programme va aller chercher les codes d'allumage les uns après les autres pour activer les segments de l'afficheur, sélectionnés à partir d'un code mis dans DISCNT.

Le listing détaillé de ce sous-programme est donné en encadré.

- L'appel du sous-programme d'affichage se fait avec l'instruction JSR \$E07B (notez l'adressage étendu).

- Mais comment l'unité centrale sait-elle où retourner dans notre programme principal lorsque le sous-programme d'affichage est exécuté ?

Cela lui est facile car elle a sauvegardé l'adresse de retour, ici 0015, après avoir décodé l'instruction d'appel du sous-programme JSR. Cette adresse, qui était alors contenue dans le compteur-programme (PC) est sauvegardée dans une zone de la mémoire RAM appelée « Pile » (« Stack » en anglais), car c'est là qu'on « empile » les données à sauvegarder. C'est dans cette pile que l'unité centrale viendra chercher l'adresse où retrouver (0015) à la fin du sous-programme pour la remettre dans le compteur-programme PC par l'instruction PULS PC ou RTS. Notez que le sous-programme d'affichage sauvegarde aussi les contenus de X, B et A en les empilant par l'instruction PSHS et les récu-

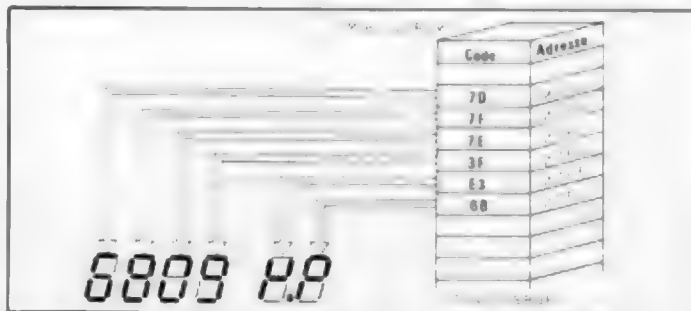


Fig. 20

père en les dépilant par l'instruction PULS, ceci afin d'éviter que leurs valeurs utilisées dans un programme principal soient modifiées.

La figure 22 schématise ces opérations de sauvegarde dans la pile.

Symbole	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Code d'allumage 7 segments	7E	06	5B	1F	27	3D	7D	0E	7F	3F	6F	75	78	57	79	69

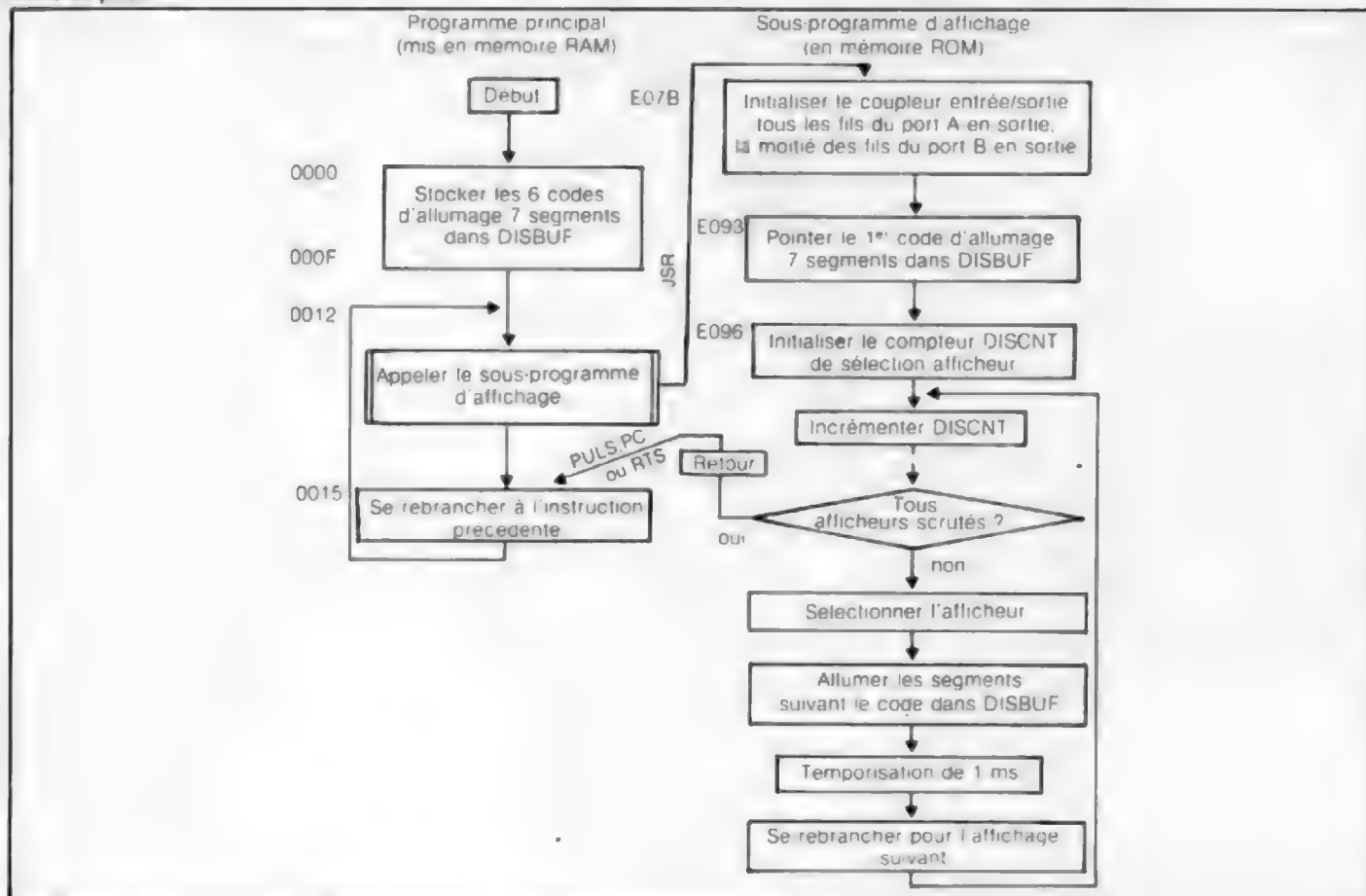


Fig. 21 : Ordinoigramme complet du programme d'affichage.

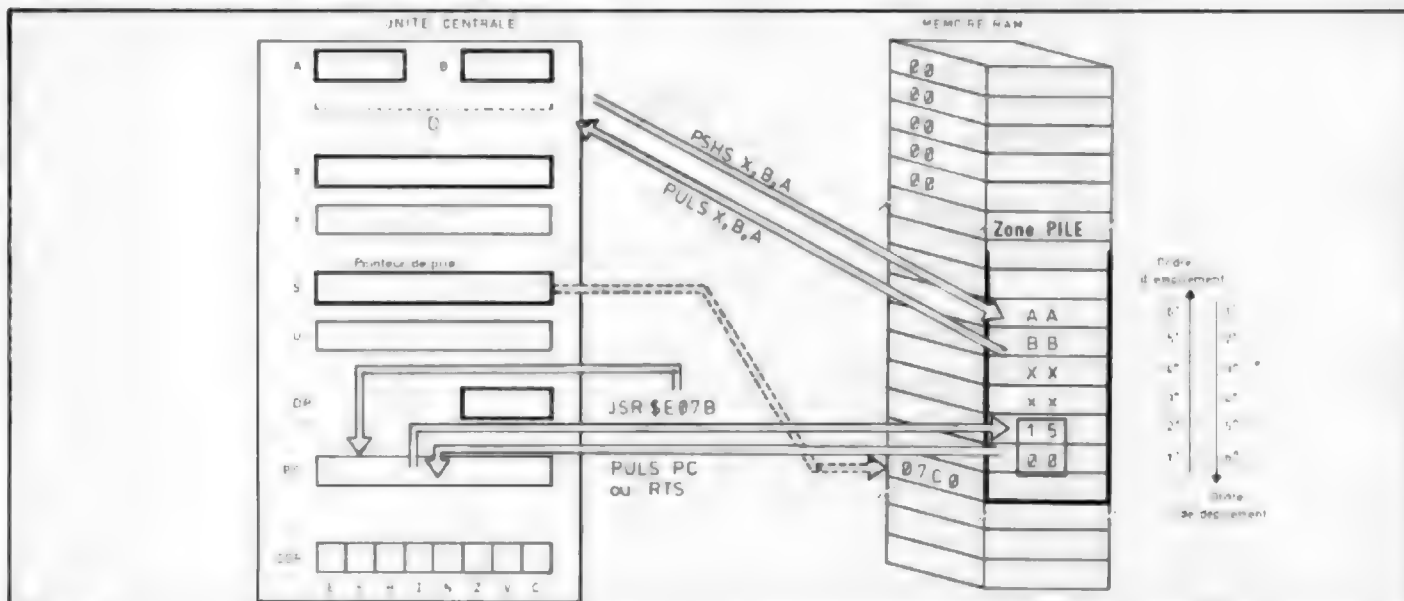


Fig. 22

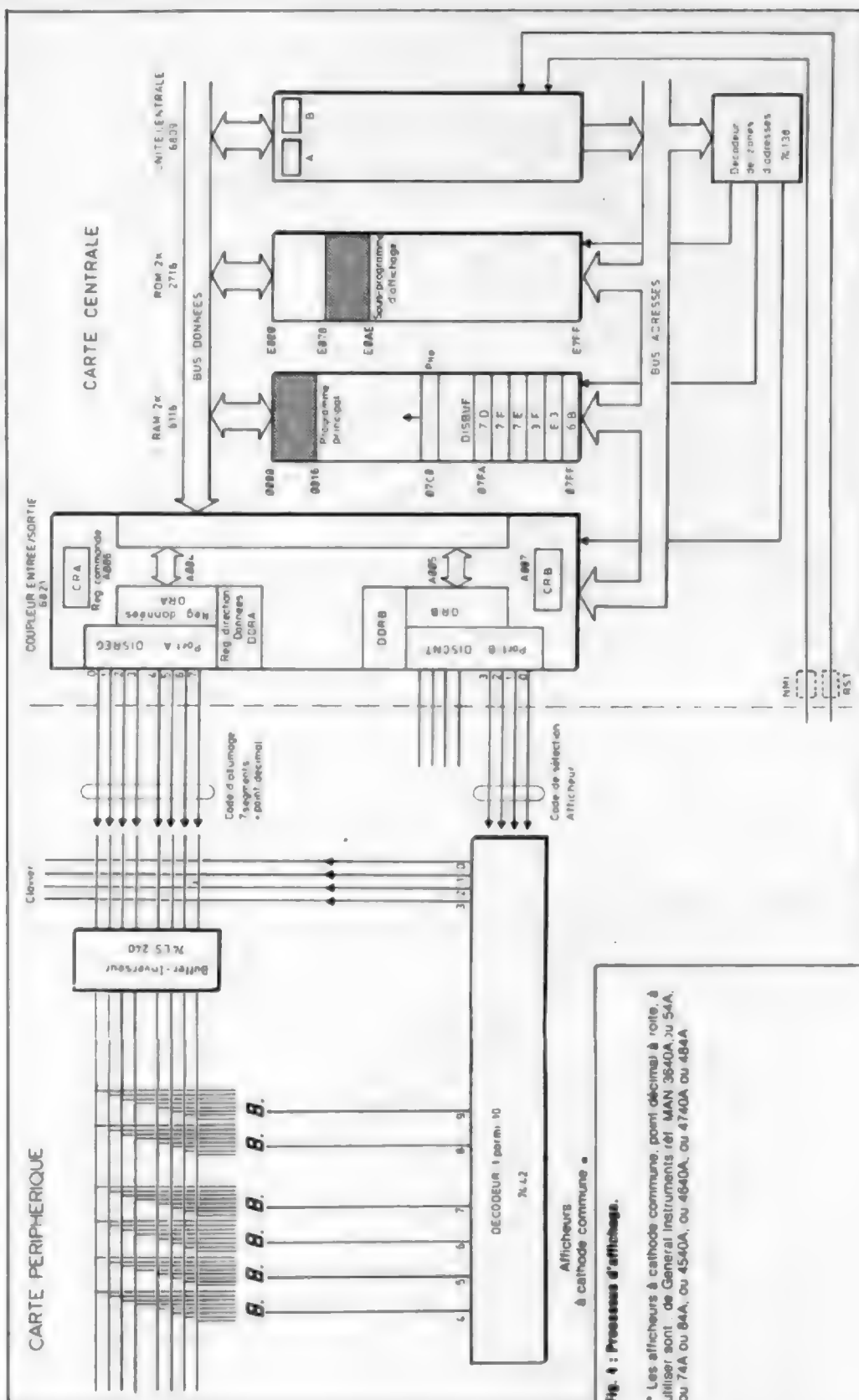


Fig. 4 : Processus d'affichage.

\* Les afficheurs à cathode commune, point décimal à roite, à utiliser sont : de General Instruments réf. MAN 3640A ou 54A, ou 74A ou 84A, ou 4540A, ou 4640A, ou 4740A ou 484A.



### Listing des programmes-exercices

					<b>Programme n° 10 :</b> Recopie de zone-mémoire
0 1 0 0	8 E 0 2 0 0		LDX	#\$0200	X pointe l'adresse de début SOURCE
0 3	A 6 8 4	LOOP	→LDA	.X	Charger la donnée pointée
0 5	A 7 0 8		STA	8.X	La stocker à l'adresse (X) + 8
0 7	3 0 0 1		LEAX	1.X	Pointer adresse suivante
0 9	8 C 0 2 0 8		CMPX	#\$0208	Fin de zone-mémoire ?
0 C	2 6 F 5		↙BNE	LOOP	Si non continuer rangement
0 E	3 F		SWI		Si oui arrêt et retour moniteur
Réponse « blanche » : utiliser l'instruction STA - 8,X					
					<b>Programme n° 11 :</b> Rangement d'octets
0 1 1 0	8 E 0 3 0 0		LDX	#\$0300	X pointe en début de table
1 3	4 F		CLRA		Initialisation du générateur d'octets
1 4	5 F		CLRB		
1 5	E 7 8 B	LOOP	→STB	D.X	Contenu de B à l'adresse (D) + X
1 7	5 C		INCB		
1 8	2 6 F B		↙BNE	LOOP	Si B ≠ 00 continuer rangement
1 A	3 F		SWI		Si B = FF arrêt et retour moniteur

**Réponse « rouge » :** La valeur du déplacement étant codée en complément à 2, un déplacement de 256 adresses (\$00 à \$FF) nécessite un nombre d'au moins huit chiffres binaires significatifs et d'un bit de signe, soit neuf chiffres. D'où nécessité d'utiliser le registre D de capacité 16 chiffres.

Si l'on utilise un accumulateur A ou B, de huit chiffres, seuls les sept premiers chiffres (les bits n° 0 à 6) sont significatifs de la valeur absolue du « déplacement », le huitième chiffre (le bit n° 7) indiquant le signe. Or le contenu de l'accumulateur varie :

— de \$00 (0000 0000 en binaire) à \$7F (0111 1111), soit de +0 à +127 en décimal,

— puis de \$80 (1000 0000 en binaire) à \$FF (1111 1111) soit de -128 à -1.

On rangera donc bien des octets d'ordre croissant de \$0500 à \$057F (\$0500 + 127) puis de \$0480 (\$0500 - 128 adresses) à \$04FF (\$0500 - 1).

		Etiquette	Nbre de cycles			<b>Programme n° 12 :</b> double décompteur
0 1 2 0	8 E F F F F		3	→LDX	#\$FFFF	Charge X avec 65 535
2 3	8 6 F F	LOOP1	2	→LDA	#\$FF	Charge A avec 255
2 5	4 A	LOOP2	2	↙DECA		Décompter
2 6	2 6 F D		3	BNE	LOOP 2	Si A ≠ 0 continuer décompt. chif. infér.
2 8	3 0 8 2		4 + 2	↙LEAX	. - X	Sinon décompter chiffres supérieurs
2 A	2 6 F 7		3	BNE	LOOP 1	Si X ≠ 0 continuer décomptage
2 C	3 F		19	SWI		Sinon arrêt et retour moniteur

Le temps total d'exécution du programme est

$$3 + [(2 + 3) 255] + 6 + 3 + 2) \times 65\,535] + 19 = 84\,278\,032 \mu s \approx 84.3 \text{ secondes.}$$

		Adresse-programme		Code-instruction		Etiquette	Langage Assembleur		Commentaires
00785	00156					##### ALLUMAGE DES AFFICHEURS #####			
00795	00158A	E07B	34	16	A	DISPRE	PSHS	X,0,A	
00800	00159A	E07D	8E	A004	A		LDI	00ISREG	
00805	00160A	E080	4F				CLRA		
00810	00161A	E081	A7	02	A		STA	2,X	ACCES A DIPA
00815	00162A	E083	A7	03	A		STA	3,X	ACCES A DORB
00820	00163A	E085	86	7F	A		LDA	007F	
00825	00164A	E087	A7	84	A		STA	,X	PA EN SORTIE
00830	00165A	E089	86	0F	A		LDA	000F	
00835	00166A	E08B	A7	01	A		STA	1,X	PBO-3 EN SORTIE
00840	00167A	E08D	86	04	A		LDA	0004	
00845	00168A	E08F	A7	02	A		STA	2,X	ACCES A PA-DISREG
00850	00169A	E091	A7	03	A		STA	3,X	ACCES A PB-DISCONT
00855	00170A	E093	8E	07FA	A		LBI	00ISBUF	
00860	00171A	E096	C6	03	A		LDB	0003	
00865	00172A	E098	5C			RECON	INCB		
00870	00173A	E099	C1	0A	A		CMPB	000A	TOUS AFFICHEURS SCRUTES?
00875	00174A	E09B	26	02	E09F		BNE	SCRUTA	NON,CONTINUER
00880	00175A	E09D	35	96	A		PULS	PC,X,0,A	OUI,RETOUR SOUS GETKEY
00890	00177					##### ALLUMER UN AFFICHEUR APRES L'AUTRE #####			
00900	00179A	E09F	F7	A005	A	SCRUTA	STB	DISCONT	CHOISIR L'AFFICHEUR
00905	00180A	E0A2	46	80	A		LDA	,X+	PRENDRE CAPACTERE DS
00910	00181A	E0A4	43				COMA		
00915	00182A	E0A5	37	A004	A		STA	DISREG	ALLUMER SEGMENTS
00920	00183A	E0A8	36	A0	A		LDA	00A0	DISBUF
00925	00184A	E0AA	4A			DLY1	DECA		
00930	00185A	E0AB	26	F0	E0AA		BNE	DLY1	DUREE D'INS
00935	00186A	E0AD	20	E9	E098		BRA	RECON	ALLUMER TS AFFICHEURS

Listing du sous-programme d'affichage.

## Chapitre 3

# Rôle des interruptions matérielles et logicielles

Ce troisième chapitre termine la première série consacrée à la présentation et à la réalisation de la maquette microkit 09 ainsi qu'à l'apprentissage des techniques de base de sa programmation. Il nous permettra de mieux comprendre et d'utiliser les interruptions du 6809 pour gérer des périphériques (PIA, ACIA, GDP, Timer, CAD, CDA, GPIA... \*) ou pour démarrer un programme à partir d'une sollicitation extérieure.

Avant tout, nous sommes en droit de nous poser la question suivante : « qu'est-ce qu'une interruption ? ».

C'est un moyen matériel (donc un signal représenté par le changement d'état d'une ligne) ou logiciel (donc une instruction placée dans le programme en cours d'exécution), qui permet :

- d'interrompre un programme en cours
- de traiter prioritairement un programme par rapport à un autre qui se trouve, par principe, moins prioritaire.
- de revenir, éventuellement, à la situation où l'on se trouvait avant la demande d'interruption ou d'attendre une autre interruption (dans ce cas, le CPU ne sera affecté qu'à une tâche de gestion d'interruption).

Le processeur peut ainsi traiter des problèmes « en temps réel »... limité seulement par sa vitesse de traitement en fonction des besoins extérieurs.

Le processeur 6809 possède un système très complet d'interruptions :

- interruptions logicielles qui viennent du programme lui-même (demande d'arrêt du programme, exécution du programme pas-à-pas pour une visualisation automatique des registres du microprocesseur, demande de lecture ou d'écriture sur un ou des organes périphériques...).

Les interruptions correspondantes s'appellent : « Software Interrupt ».

SWI  
SWI2  
SWI3

- Interruptions matérielles qui sont au nombre de 3 :
  - NMI (Non Maskable Interrupt) : interruption non masquable
  - FIRQ (Fast Interrupt Request) : demande d'interruption rapide
  - IRQ (Interrupt Request) : demande d'interruption
- Mise en attente ou synchronisation sur un événement extérieur dont la présence est signalée par une ou plusieurs

entrées d'interruptions : ce sont les instructions.

- CWA (Clear and Wait Interrupt) : attente d'interruption...

Nous verrons plus loin à quoi sert le Clear.

- SYNC (attente d'une synchronisation externe).

Pour corser le tout, citons l'existence de 2 broches (fig. 1) Halt et Dma/Breq servant à déconnecter le microprocesseur de son environnement afin de permettre des traitements plus spécialisés.

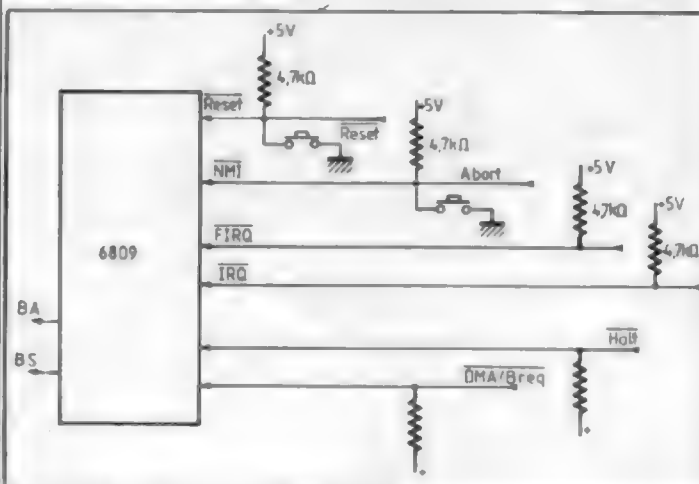


Fig. 1 : Les lignes d'interruptions matérielles.

Afin de nous dire où il en est de ses pérégrinations, le microprocesseur positionne deux lignes de sortie BA et BS selon 4 états possibles : (fig. 2).

BA= Bus disponible	BS= état du III	états du CPU
0	0	fonctionnement normal
0	1	Interruption quelconque reconnue
1	0	SYNC acceptée
1	1	Halt ou Bus accordé

BA = Bus Available

BS = Bus Status

Fig. 2 : Table des états du CPU.

\* ACIA : Asynchrone Communication Interface Adapter = Périphérique d'entrée-sortie série ; GPIA = General Purpose Interface Adapter = Interface d'applications générales (Bus IEEE, IEC) ; GDP = Graphic Display Processor = Processeur graphique.

Les lignes d'interruptions sont toujours actives à l'état bas et les entrées du CPU sont à collecteur ouvert, ce qui permet de relier plusieurs périphériques sur la même ligne constituant ainsi un «ou câblé».

Afin de permettre une bonne gestion sans confusion des interruptions, l'unité de séquençement du microprocesseur est programmée pour les exécuter en considérant leur priorité respective ; depuis la plus importante (celle qu'il faut exécuter avant toute autre) jusqu'à la plus faible (celle qu'il faut exécuter après toute autre).

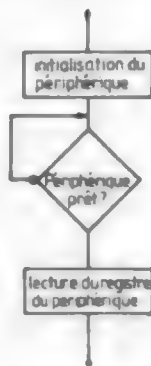
Les priorités sont les suivantes :



Ainsi, si  $\overline{\text{FIRQ}}$  et  $\overline{\text{IRQ}}$  passent simultanément à l'état bas, c'est  $\overline{\text{FIRQ}}$  qui sera la première prise en compte.

Remarquons que SWI2 et SWI3 possèdent le même niveau de priorité ; ces deux interruptions sont utilisées par des logiciels d'aide au développement (comme UNIX ou OS9) pour assurer un «service request» et facilite ainsi grandement la programmation.

Par exemple, s'il faut lire le contenu du registre d'un périphérique, il faut écrire, par la méthode traditionnelle, des lignes de programme comme le montre l'organigramme ci-contre :



Par le «service request», il suffit tout simplement d'écrire un code qui pourrait être : OS9 I\$READ.

Le logiciel interprète ce code comme un SWI2 suivi d'une adresse qui lui permet de faire l'exécution demandée et de revenir au programme principal.

De toute façon, quelle que soit l'interruption demandée, le microprocesseur doit :

- interrompre le programme principal (ou en cours)
- garder tout ou partie du contexte dans une pile
- exécuter une séquence privilégiée, reflet du type de traitement d'interruption
- prendre en compte l'interruption toujours après l'exécution complète d'une instruction (sinon, «bonjour les dégâts !»).

Le microprocesseur doit savoir, à tout moment, où se brancher pour exécuter l'interruption demandée.

Il dispose ainsi de 14 adresses mémoires comprises entre \$FFF2 et \$FFFF qui lui permettront de savoir où se brancher pour exécuter l'interruption désirée ; en outre, la pile S sauvegarde tous les registres en mémoire, y compris le compteur de programme, pour permettre au CPU de revenir au programme qui était en cours d'exécution avant l'interruption.

Le programmeur doit être très vigilant sur la gestion de cette pile, la moindre erreur est souvent lourde de conséquences.

Les cases mémoires comprises entre \$FFF2 et \$FFFF sont affectées à des adresses représentatives des interruptions (fig. 3) hard et soft.

Ces adresses s'appellent des «vecteurs d'interruptions».

Ainsi, le vecteur d'interruption du Reset se trouve en \$FFFE et \$FFFF.

FFFE/FFFF	RESET	LSB MSB
FFFC/FFFD	NMI	LSB MSB
FFFA/FFFB	SWI	LSB MSB
FFF8/FFF9	IRQ	LSB MSB
FFF6/FFF7	FIRQ	LSB MSB
FFF4/FFF5	SWI2	LSB MSB
FFF2/FFF3	SWI3	LSB MSB
FFF0/FFF1	réserve par le CPU	LSB MSB

Fig. 3 : Table des vecteurs d'interruptions.

A cette adresse, le CPU va trouver une adresse qu'il placera dans son compteur de programme, ce qui lui permettra de se brancher au programme de reset demandé. Ce qui revient à dire que le microprocesseur va chez Dupont (adresses \$FFFE et \$FFFF) demander l'adresse de Durant (le vecteur qui se trouve en \$FFFE et \$FFFF).

Il s'agit d'un adressage étendu indirect représenté par le mnémonique : JMP [\$FFFE] codé par 6E 9F FF FE (fig. 4). Aidez-vous des tableaux d'instructions présentés dans le

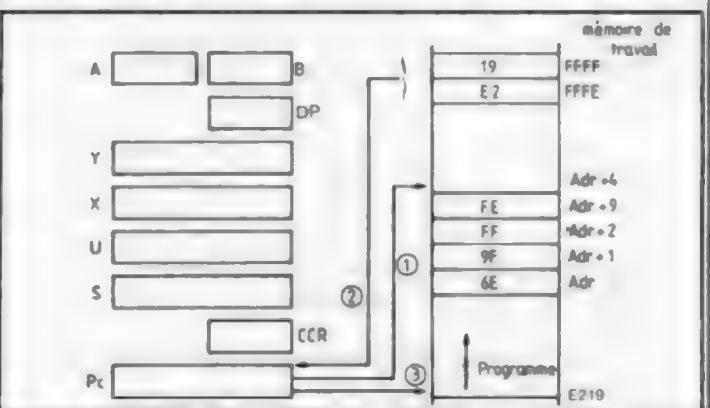


Fig. 4 : Séquençement des opérations de JMP [\$FFFE].

① Le PC pointe sur Adr + 4 (instruction suivante).

② Chargement du vecteur E219 dans PC.

③ soit en E219 pour exécuter la routine demandée.

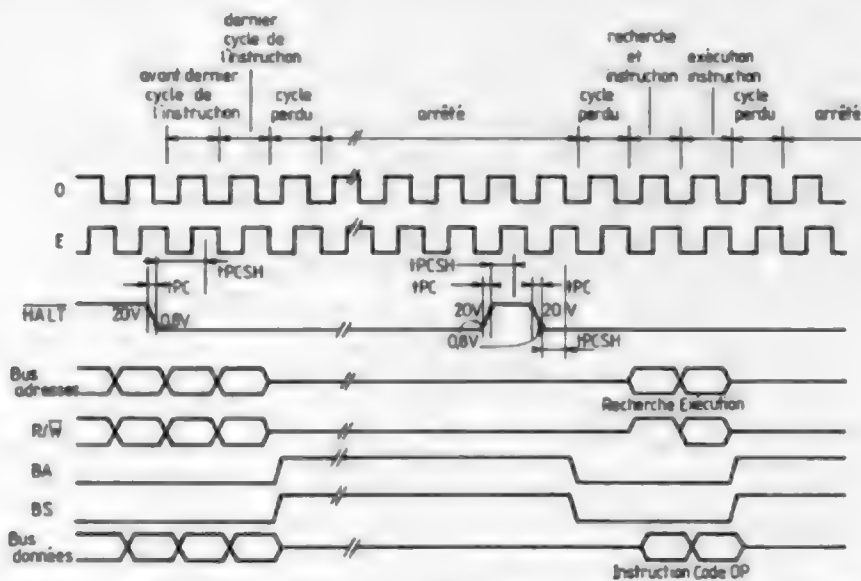


Fig. 5 : Mode Halt et exécution d'une seule instruction (tiré des documents-EFCIS).

chapitre deux pour retrouver le code opératoire.

Bien entendu, le microprocesseur effectue automatiquement ce branchement lorsqu'il rencontre une instruction d'interruption ou lorsqu'une de ses lignes hard passe à 0.

Revenons à l'explication des lignes Halt et Dma/Breq :

\* Halt (fig. 5) : à l'état haut, le microprocesseur est en fonctionnement normal ( $BA = BS = 0$ ), c'est le cas du microkit 09. A l'état bas, le microprocesseur s'arrête ( $BA = BS = 1$ ) à la fin de l'instruction en cours et demeure à l'arrêt sans perte de données puisque l'horloge continue à fonctionner normalement pour rafraîchir les registres internes du CPU.

Le microprocesseur est donc en mode «off», ses bus d'adresses et de données ainsi que la ligne R/W étant à l'état haute impédance.

Un passage à 1, d'une durée d'un cycle, redémarre le CPU qui n'exécute qu'une seule instruction pour se mettre ensuite en mode off.

L'utilisation intelligente de cette ligne permet de travailler en mode multi-processeurs (fig. 6) : une unité centrale appelée «unité maître» assure l'aiguillage des tâches des unités asservies appelées «unités esclaves».

Les micro-ordinateurs modernes présentent souvent cette configuration : une unité centrale assurant la scrutation d'un clavier alphanumérique, une autre unité assurant l'affichage sur un moniteur vidéo, ..., l'unité maître assurant l'aiguillage et la gestion des tâches.

\* Dma/Breq : Direct Memory Access/Bus Request (fig. 7) : cette ligne peut avoir deux utilisations différentes :

- accès direct à la mémoire
- rafraîchissement de mémoires dynamiques

Dans le premier cas, un circuit spécialisé appelé DMAC (Direct Memory Access Controller) fait une demande d'accès au bus en mettant à 0 la broche Dma/Breq du CPU ; ce dernier transfère le contrôle au DMAC en mettant ses lignes  $BA = BS = 1$ .

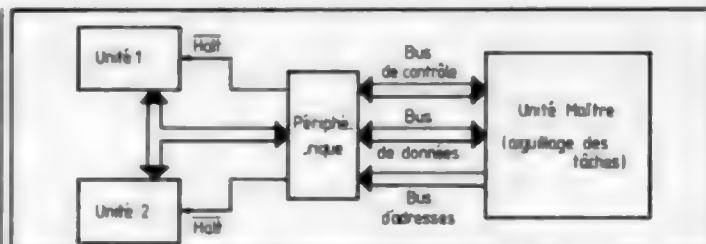


Fig. 6 : Configuration multi-processeurs.

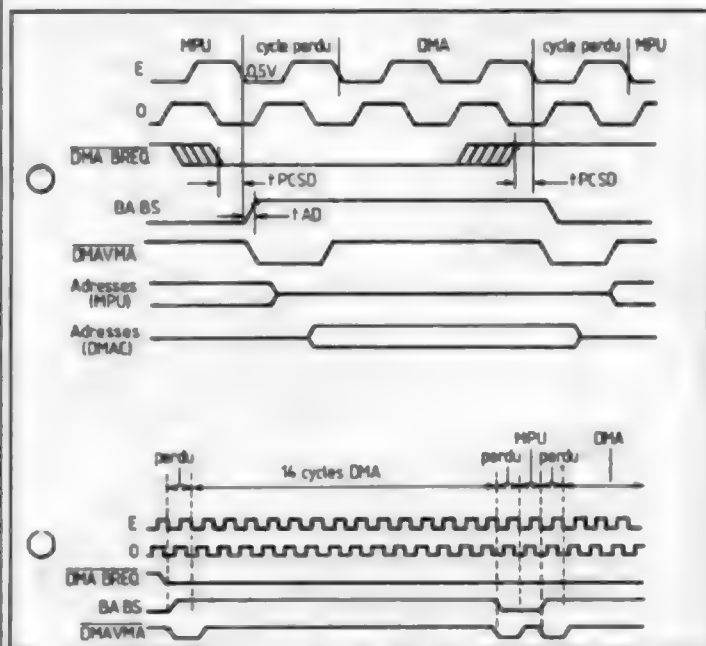


Fig. 7 : Diagramme des temps de l'entrée DMA/BREQ (documents FFCIS).

1) Accès direct à la mémoire.

2) Auto-rafraîchissement en DMA.



Durant tout le temps où  $BA = BS = 1$ , le bus d'adresses se trouve en haute impédance, ce qui permet au DMAC de se charger de la gestion de ce bus.

Il faut remarquer qu'un cycle est perdu lorsqu'on accède au DMA et lorsqu'on rend la main au microprocesseur, d'où l'utilité de fabriquer un signal  $\overline{DMAVMA}$  qui tient compte de ce fait : on exécute seulement lorsque  $\overline{DMAVMA} = 1$ .

L'accès direct à la mémoire est utilisé pour accéder rapidement à des données se trouvant à l'extérieur d'un micro-ordinateur (en général une mémoire de masse telle qu'un lecteur de disquettes) et pour effectuer un chargement rapide dans la mémoire RAM du micro-ordinateur (fig. 8).

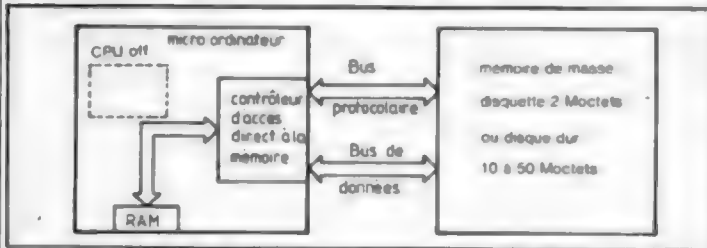


Fig. 8 : Processus de DMA.

Dans le deuxième cas, on arrête le CPU pour rafraîchir des mémoires dynamiques.

Au-delà de 14 cycles, le CPU doit reprendre la main pour auto-rafraîchissement.

Ce type de fonctionnement est de moins en moins utilisé car il oblige à bloquer le CPU, ce qui fait perdre du temps ; or, les mémoires dynamiques ont aujourd'hui des temps d'accès très courts, ce qui permet de les rafraîchir dans un temps beaucoup plus court.

On remarque que des données ne se trouvent jamais sur le bus de données lorsque  $E=0$  : il est donc possible de rafraîchir une case mémoire durant ce court laps de temps... sans arrêt du CPU.

Nous aurons l'occasion de parler de tout cela lors de la réalisation du microcomp... l'année prochaine.

## LES INTERRUPTIONS

Nous allons maintenant décrire les interruptions du 6809 (flow chart de la fig. 9).

### Interruptions matérielles

**Le Reset** (fig. 10) : il s'agit de l'interruption la plus prioritaire et c'est bien normal puisqu'il faut bien dire au CPU par quoi commencer lors de la mise sous tension.

D'autre part, si un programme se « plante » (impossible de reprendre la main) une action sur le bouton reset permet de reprendre le contrôle.

Enfin, le programme de reset permet de charger les registres à des valeurs spécifiques nécessaires pour un programme donné (par exemple : initialisation de la pile  $S = \$07C0$ ) sauvegarde de valeurs constantes en RAM etc...

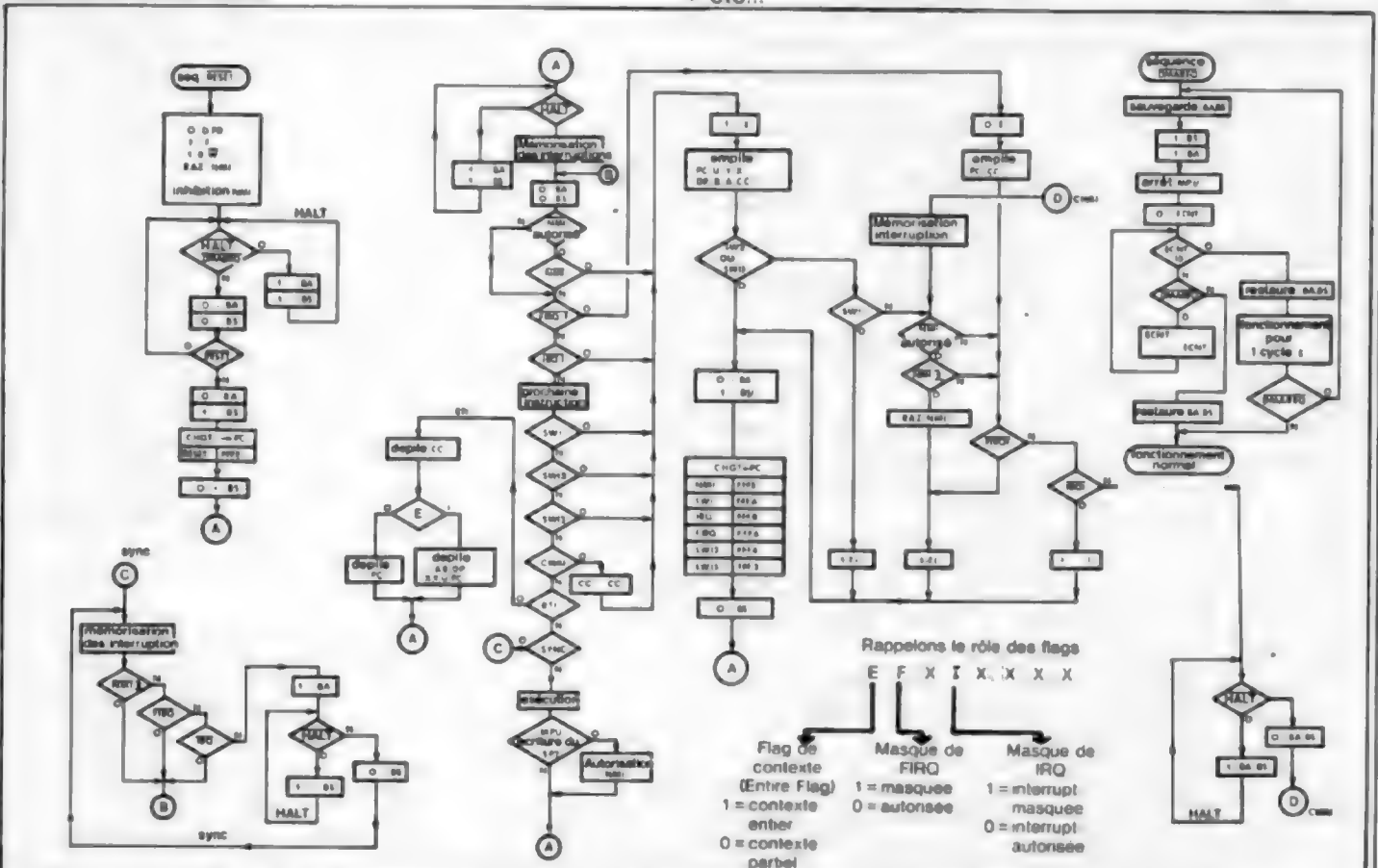


Fig. 9 : Organigramme du 6809.

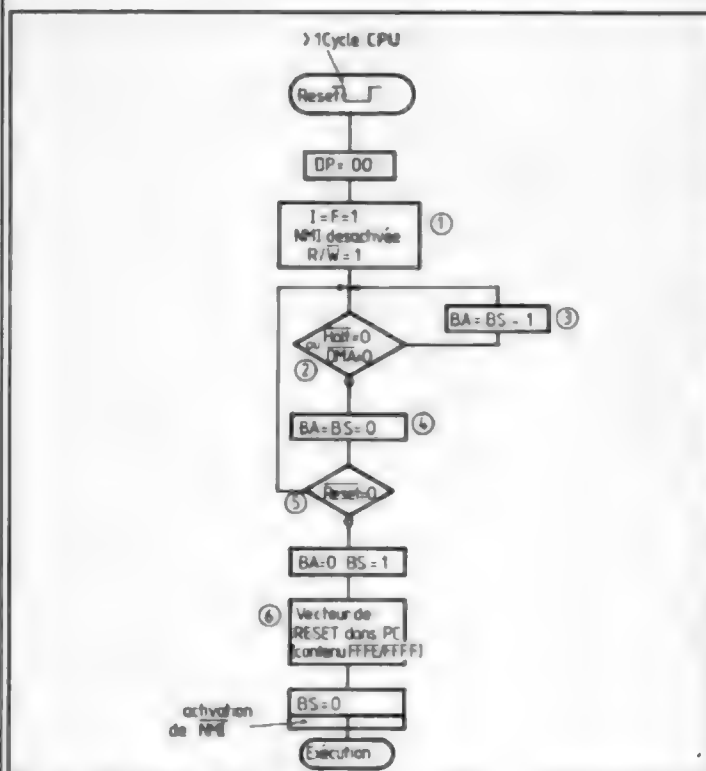


Fig. 10 : Séquence de Reset.

Cette séquence interne au CPU s'exécute en une dizaine de cycles.

Dès la mise sous tension, ou lors d'un appui sur la touche Reset, le CPU :

1. Charge le registre de page DP à 0 afin de se rendre compatible avec le 6800.

- masque toute interruption telles que  $\overline{\text{FIRQ}}$ ,  $\overline{\text{IRQ}}$  et  $\overline{\text{NMI}}$  (inutile pour SWI, SWI2 et SWI3 puisqu'il s'agit d'instructions se trouvant dans un programme).

- se met en lecture  $\text{R}/\overline{\text{W}} = 1$

2. Teste ses lignes Halt et DMA/Breq

3. Si 0 positionne les bus en haute impédance ( $\text{BA} = \text{BS} = 1$ ) et attend un retour à la normale.

4. Dans ce cas,  $\text{BA} = \text{BS} = 0$  sinon retour en 2.

5. Teste si  $\overline{\text{Reset}} = 0$ , (en effet, une action sur la touche Reset dure sûrement plus longtemps qu'un cycle CPU = 1  $\mu\text{s}$ ..., nous sommes beaucoup moins rapides que le microprocesseur, on s'en doute !).

6. Dès que la touche est relâchée, on se met en mode reconnaissance d'interruption ( $\text{BA} = 0$  et  $\text{BS} = 1$ ) pour aller chercher le vecteur de Reset en FFFE/FFFF. Le contenu de ces deux cases est mis dans le compteur des programmes, puis le CPU se met en mode normal  $\text{BA} = \text{BS} = 0$  pour exécuter le programme.

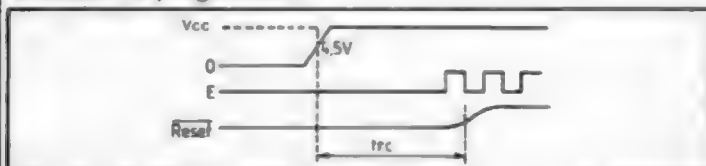


Fig. 11 : Auto Reset.

Notons qu'à la mise sous tension, l'horloge ne se met en route qu'au bout d'un temps  $t_{\text{rc}} \approx 100 \text{ ms}$ , le Reset n'est pris en compte qu'un cycle plus tard (fig. 11).

## L'interruption NMI : (fig. 12)

Cette ligne d'interruption non masquable ne peut être ignorée (masquée) par le microprocesseur, elle est donc d'un niveau plus élevé que les autres interruptions ( $\overline{\text{IRQ}}$ ,  $\overline{\text{FIRQ}}$ ) mais moins élevée que le Reset qui la désactive. Si le microprocesseur est à l'arrêt, un front actif sur  $\overline{\text{NMI}}$  sera mémorisé pour une réponse différée.

Puisque cette interruption est la plus prioritaire, on la réserve aux traitements devant résulter d'une défaillance d'alimentation (sauvegarde dans une mémoire C-MOS alimentée par batterie par exemple), ou pour visualiser le contenu des registres du CPU lorsqu'un programme «se plante» (bouton Abort sur le Microkit 09 et sur la plupart des systèmes de mise au point).

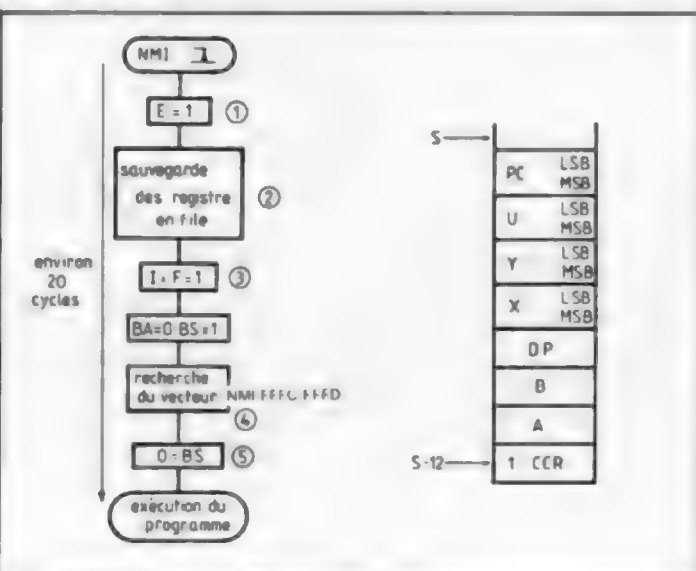


Fig. 12 : Organigramme du  $\overline{\text{NMI}}$ .

Un front actif (négatif) sur l'entrée  $\overline{\text{NMI}}$  du CPU provoque le déroulement de la séquence suivante :

① Le flag E du CCR se positionne à 1 pour indiquer que le microprocesseur sauve tous ses registres en Pile ②. On retiendra que la Pile pointe toujours «au-dessus» de ce qu'elle va ranger et pointe toujours sur ce qu'elle vient de ranger.

Par exemple : avant sauvegarde  $\text{S} = \$07\text{C0}$

après sauvegarde S pointe sur le dernier registre rangé (CCR) en  $\$07\text{B4}$ .

③ Le CPU masque les interruptions  $\overline{\text{FIRQ}}$  et  $\overline{\text{IRQ}}$  afin qu'elles ne soient pas exécutées durant le déroulement du programme de NMI. Puis il se met en mode reconnaissance de l'interruption et va chercher le vecteur qui se trouve en  $\$FF\text{FC}/\$FF\text{FD}$  pour le mettre dans le compteur de programme ④.

A ce stade, il se met en mode exécution  $\text{BS} = \text{BA} = 0$  pour exécuter le programme de NMI (figure 13).

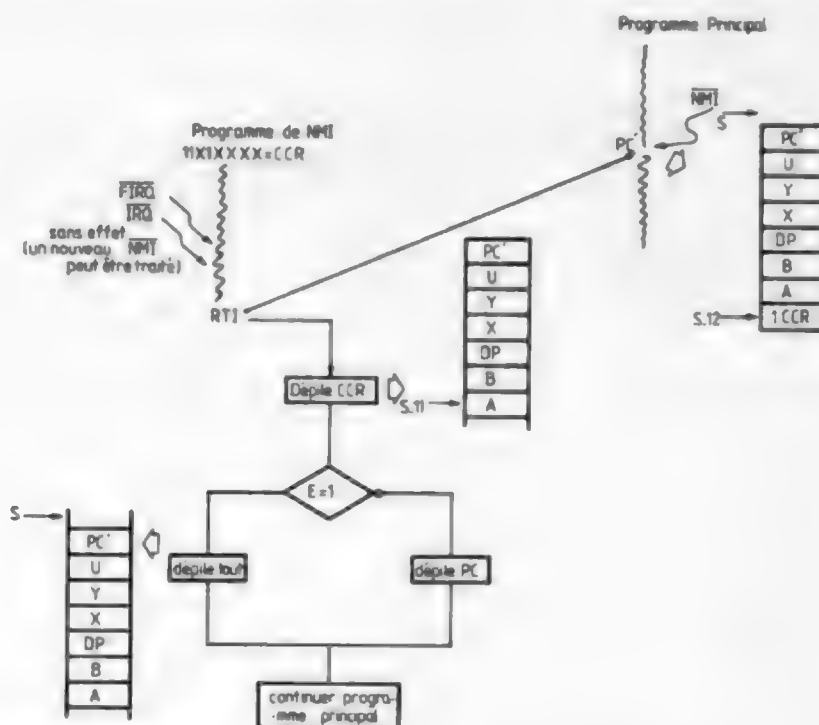


Fig. 13 : Exemple de déroulement d'un programme de NMI.

«Comment revient-on au programme principal ?» direz-vous. Ceci est bien simple, on insère, en fin de programme d'interruption, une instruction RTI (Return From Interrupt) qui permet de dépiler la pile S pour remettre le CPU dans l'état où il était avant interruption (fig. 14).

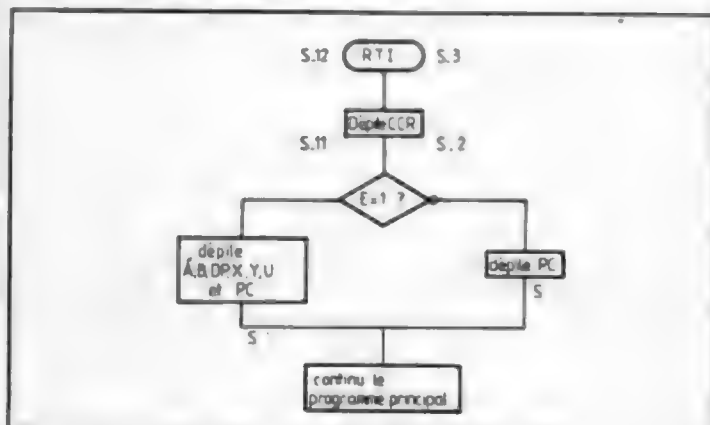


Fig. 14 : Organigramme de l'instruction RTI.

## L'interruption $\overline{\text{FIRQ}}$ (fig. 15)

Cette interruption  $\overline{\text{FIRQ}}$  est masquable par l'intermédiaire du bit 5 du CCR (flag F). Elle est plus prioritaire que  $\overline{\text{IRQ}}$  puisque le CPU met un masque d'interruption sur le flag I du CCR.

L'utilisateur peut sauvegarder dans la pile d'autres registres que PC et CCR grâce à l'instruction PSHS.

Signalons au passage l'utilité des instructions ANDCC et ORCC qui s'utilisent exclusivement en adressage immédiat :

- ANDCC # \$xx effectue le ET entre (CCR) et xx et place le résultat dans CCR.

Cette instruction positionne donc un bit particulier à 0

Ex. 1

	E	F	H	J	N	Z	V	C
CCR =	0	1	1	0	1	1	1	1
xx =	1	1	1	1	0	1	1	1
nouveau CCR =	0	1	1	0	0	1	1	1

flag a mettre à 0  
recopie  
imposé à 0 par le ET logique

Ex 2 pour I = 0

CCR =	1	1	1	1	0	1	0	1
xx =	1	1	1	0	1	1	1	1
nouveau CCR =	1	1	1	0	0	1	0	1

$\text{ANDCC \# \$EF}$

Ex. 3 pour F = 0

CCR =	1	1	1	1	0	1	0	1
xx =	1	0	1	1	1	1	1	1
nouveau CCR =	1	0	1	1	0	1	0	1

$\text{ANDCC \# \$BF}$



• ORCC # \$xx effectue le OU logique entre (CCR) et xx et place le résultat dans CCR.  
 Cette instruction positionne donc un bit particulier à 1.

Ex. 1 pour I = 1

	E	F	H	I	N	Z	V	C	flag à mettre à 1
CCR =	0	1	1	0	1	1	0	1	
xx =	0	0	0	1	0	0	0	0	= \$ 10
nouveau CCR =	0	1	1	1	1	1	0	1	recopié

↑ imposé à 1 par le OU logique

Ex. 2 pour F = 1

	E	F	H	I	N	Z	V	C	
CCR =	0	0	0	0	1	0	1	0	
xx =	0	1	0	0	0	0	0	0	= \$ 40
nouveau CCR =	0	1	0	0	1	0	1	0	ORCC # \$40

## L'Interruption IRQ (fig. 16).

C'est l'interruption matérielle la moins prioritaire car le flag F n'est pas positionné à 1, on peut la masquer par l'intermédiaire du bit 4 du CCR (flag I)

### Remarques sur les interruptions hard

1. Il faut éviter d'envoyer une seconde interruption  $\overline{\text{NMI}}$  avant la fin du traitement de la première car  $\overline{\text{NMI}}$  est toujours prise en compte !

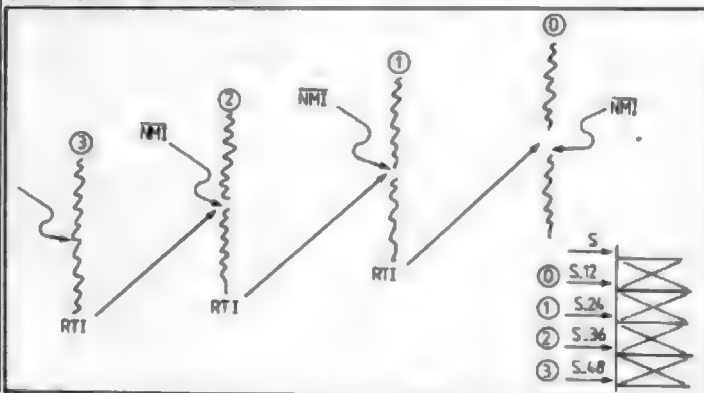


Fig. 17 : Gare aux NMI en cascade !

Si une  $\overline{\text{NMI}}$  survient régulièrement avant la fin du traitement de l'interruption initiale, on arrive à un débordement de Pile (fig. 17).

2. Durant le traitement d'une interruption  $\overline{\text{FIRQ}}$ , si l'entrée  $\overline{\text{FIRQ}}$  a retrouvé son état initial ( $\overline{\text{FIRQ}} = 1$ ), une nouvelle interruption  $\overline{\text{FIRQ}}$  peut survenir. Elle sera mémorisée, son traitement suivra celui de la première interruption. Il en est de même pour  $\overline{\text{IRQ}}$ . On peut donc conclure que  $\overline{\text{FIRQ}}$  et  $\overline{\text{IRQ}}$  sont mémorisables.

## Les interruptions logicielles

### SWI (SoftWare Interrupt) fig. 18

Cette instruction, dans un programme, impose l'arrêt de son exécution.

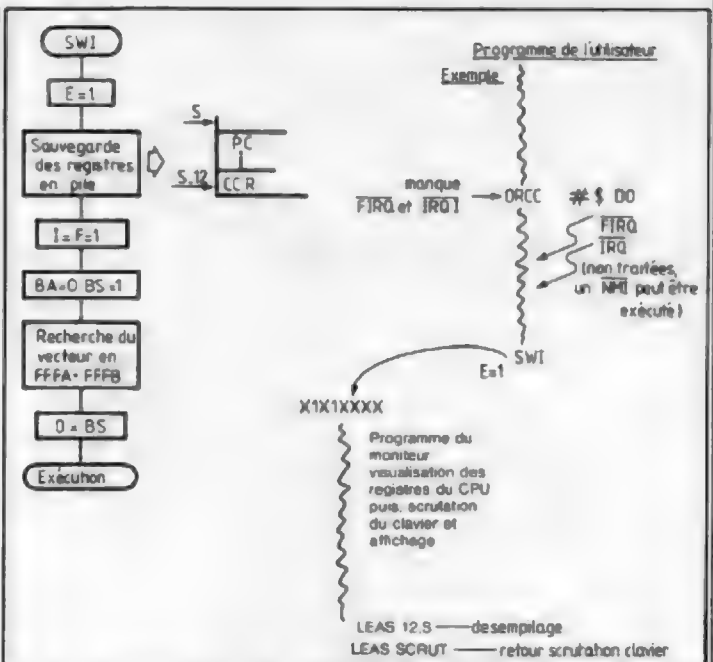


Fig. 18 : Organigramme de SWI.

Cette interruption est en générale réservée aux logiciels systèmes (par exemple arrêt d'un programme pour une visualisation automatique des registres du CPU). On remarquera que SWI est plus prioritaire que  $\overline{\text{FIRQ}}$  et  $\overline{\text{IRQ}}$  car son traitement entraîne le masquage de celle-ci.

### SWI2, SWI3 : fig. 19

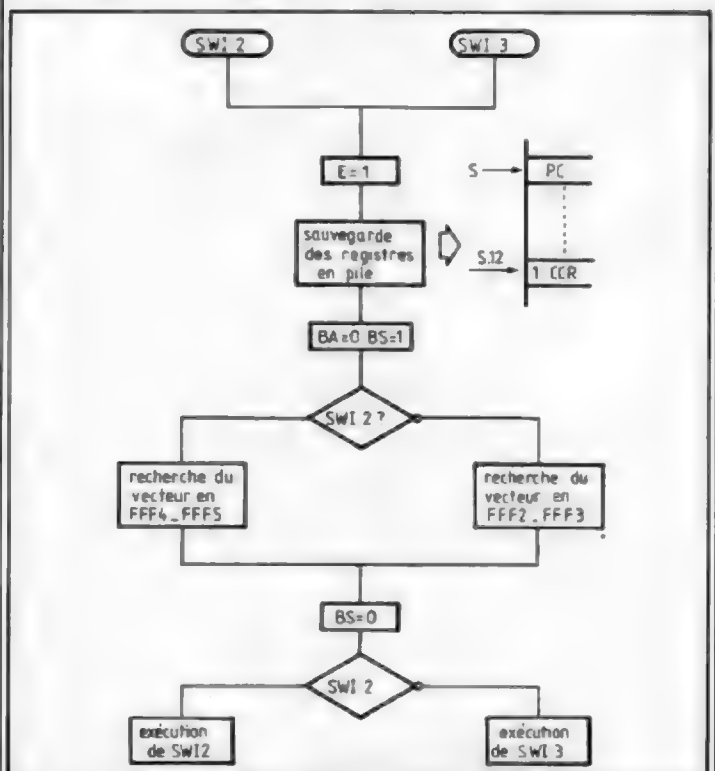


Fig. 19 : Organigramme de SWI 2 et SWI 3.



Ces deux interruptions ont un fonctionnement identique, elles peuvent être interrompues par toutes les autres interruptions du CPU.

Leur fonctionnement est identique à celui de SWI, seuls les masques d'interruption ne sont pas positionnés.

## Les instructions d'interruptions

**CWAI** (Clear and Wait Interrupt : attente d'interruption) fig. 20 :

Cette instruction, qui occupe deux octets, joue deux rôles :  
(1) Elle effectue le ET logique entre le contenu du CCR et une valeur immédiate. Le but à atteindre étant le même que l'instruction  $\text{ANDCC} \# \$ \times \times$  : mettre à 0 un flag particulier (en l'occurrence I et/ou F) d'où permission d'une interruption  $\overline{\text{IRQ}}$  et/ou  $\overline{\text{FIRQ}}$ .

(2) Arrêter le CPU qui ne démarrera que lorsqu'une interruption viendra.

Cette attente d'interruption met le CPU en veille mais non en haute impédance, BA et BS restent à zéro durant l'attente. Les registres internes sont toujours rafraichis par l'horloge du CPU.

- (1) CWAI arrête l'exécution du programme
- (2) Le CPU valide ou masque les interruptions  
 $\text{CCR} = \text{FF} - \text{IRQ}$  et  $\text{FIRQ}$  masquées
- $\text{EF} \leftarrow \text{IRQ}$  autorisée

$\text{BF} \leftarrow \text{FIRQ}$  autorisée

$\text{AF} \leftarrow \text{IRQ}$  et  $\text{FIRQ}$  autorisées

- (3)  $\text{E} = 1$  indique la sauvegarde totale du contexte du CPU
- (4) Tous les registres internes sauf S sont sauves dans la pile système.

(5) Le CPU se met en attente d'une interruption  $\overline{\text{IRQ}}$ ,  $\overline{\text{FIRQ}}$ , (suivant le contenu du CCR) ou NMI.

Remarque : Lorsqu'une interruption survient, aucun autre état du CPU n'est sauvegardé avant la vectorisation du sous-programme de traitement de l'interruption. On peut donc utiliser l'interruption  $\overline{\text{FIRQ}}$  avec une sauvegarde totale du contexte du microprocesseur (il s'agit d'un cas particulier à retenir).

## SYNC (SYNchronisation) fig. 21 :

Il s'agit d'une instruction très puissante qui ne propose pas moins de 8 possibilités différentes !

Elle permet de synchroniser le déroulement du programme sur un événement extérieur grâce aux lignes d'interruptions.

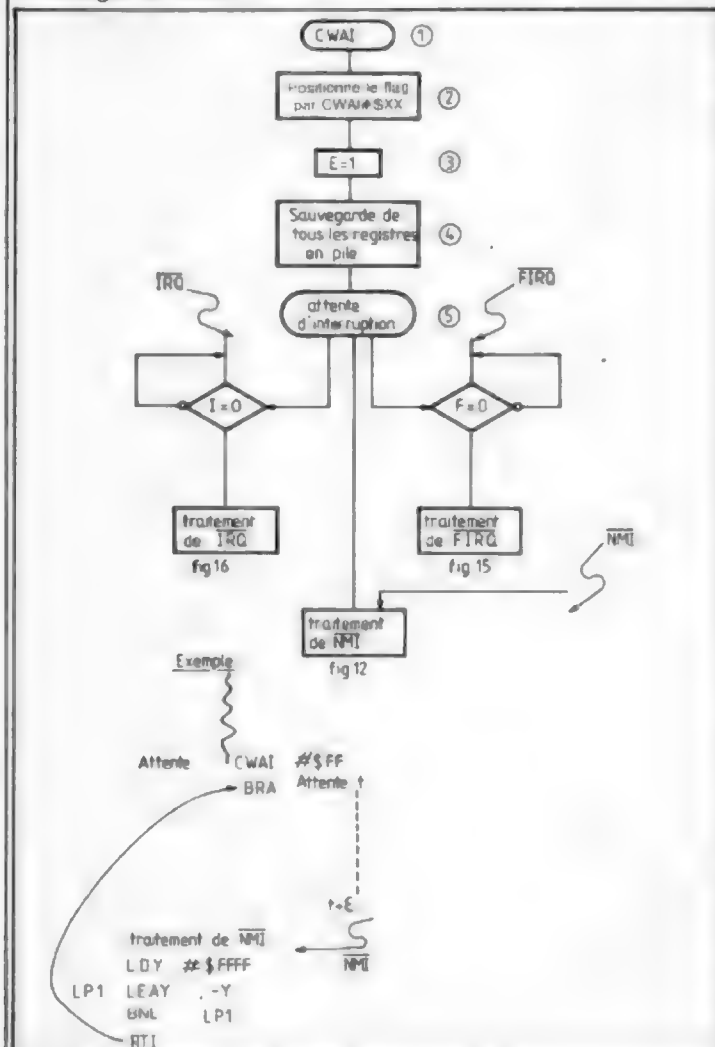


Fig. 20 : Organigramme de CWAI # \$XX.

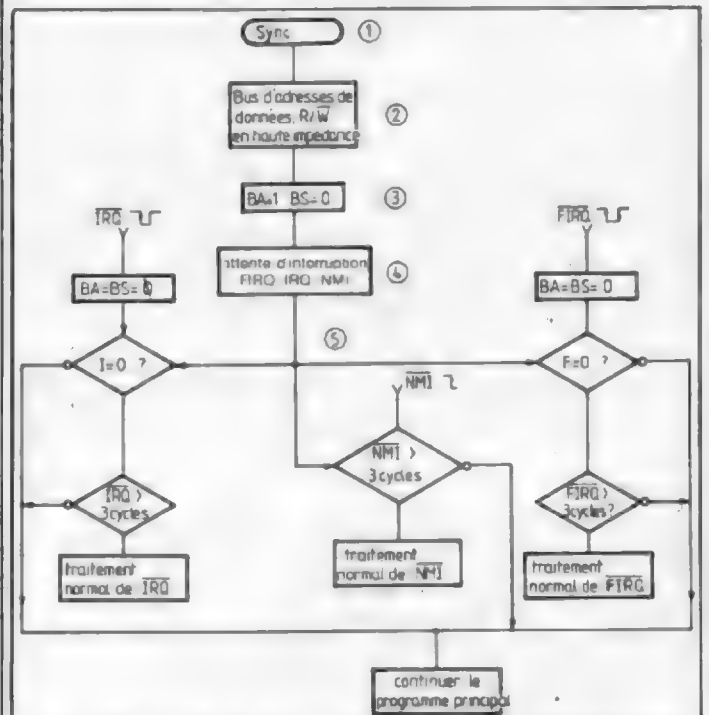


Fig. 21 : Organigramme de l'instruction SYNC.

- (1) L'instruction SYNC arrête le CPU
- (2) Les bus de données, d'adresses et la ligne R/W se mettent en haute impédance
- (3) Le CPU indique qu'il est en attente de synchronisation
- (4) Il reste dans cet état tant qu'il n'a pas reçu d'interruption  $\overline{\text{NMI}}$ ,  $\overline{\text{FIRQ}}$ ,  $\overline{\text{IRQ}}$

(5) Si une interruption se présente, sur  $\overline{IRQ}$  par exemple (fig. 22), le fonctionnement redevient normal ( $BA = BS = 0$ ).

Si  $\overline{IRQ}$  est valide, et si le niveau bas dure plus de 3 cycles CPU, celui-ci exécute le traitement approprié.

Au retour d'interruption, le processeur reprend le déroulement normal du programme.

Si l'interruption est masquée ou si le signal dure moins de 3 cycles CPU, le processeur continue le programme principal sans traiter l'interruption (le cas est le même pour  $\overline{FIRQ}$ ).

Pour  $\overline{NMI}$ , il n'est pas nécessaire de tester son flag puisqu'il n'y en a pas.

Remarque : puisqu'il est possible de mettre le CPU en haute impédance, il est facile de conclure que l'instruction SYNC peut être utilisée pour assurer des synchronisations rapides avec des périphériques, cette méthode permet éventuellement d'éviter l'utilisation d'un circuit d'accès direct à la mémoire (DMA).



Fig. 22 : Timing de l'instruction SYNC.

Note 1 : Si le bit de masque est à 1 lors d'une demande d'interruption, le traitement continue. Si une interruption non masquable ou une interruption non masquée provoquée par  $\overline{FIRQ}$  ou  $\overline{IRQ}$  est acceptée, l'adresse positionnée sur le bus depuis le cycle précédent ( $M + 1$ ) demeure sur le bus et le traitement continue avec ce cycle comme ( $M + 1$ ) du chronogramme d'interruption  $\overline{FIRQ}$ ,  $\overline{IRQ}$  ou  $\overline{NMI}$ .

Note 2 : Si les bits de masques sont mis à 0,  $\overline{IRQ}$  et  $\overline{FIRQ}$  doivent être maintenus à l'état bas bien qu'un cycle seulement soit nécessaire pour mettre le processeur hors de l'état SYNC.

### Exercice

Notre but est d'afficher le nombre d'appuis sur la touche  $\overline{NMI}$ .

Nous allons utiliser deux méthodes :

- 1) avec utilisation de la fonction SYNC
- 2) avec affichage du nombre d'appuis

L'organigramme revêtira la forme suivante :

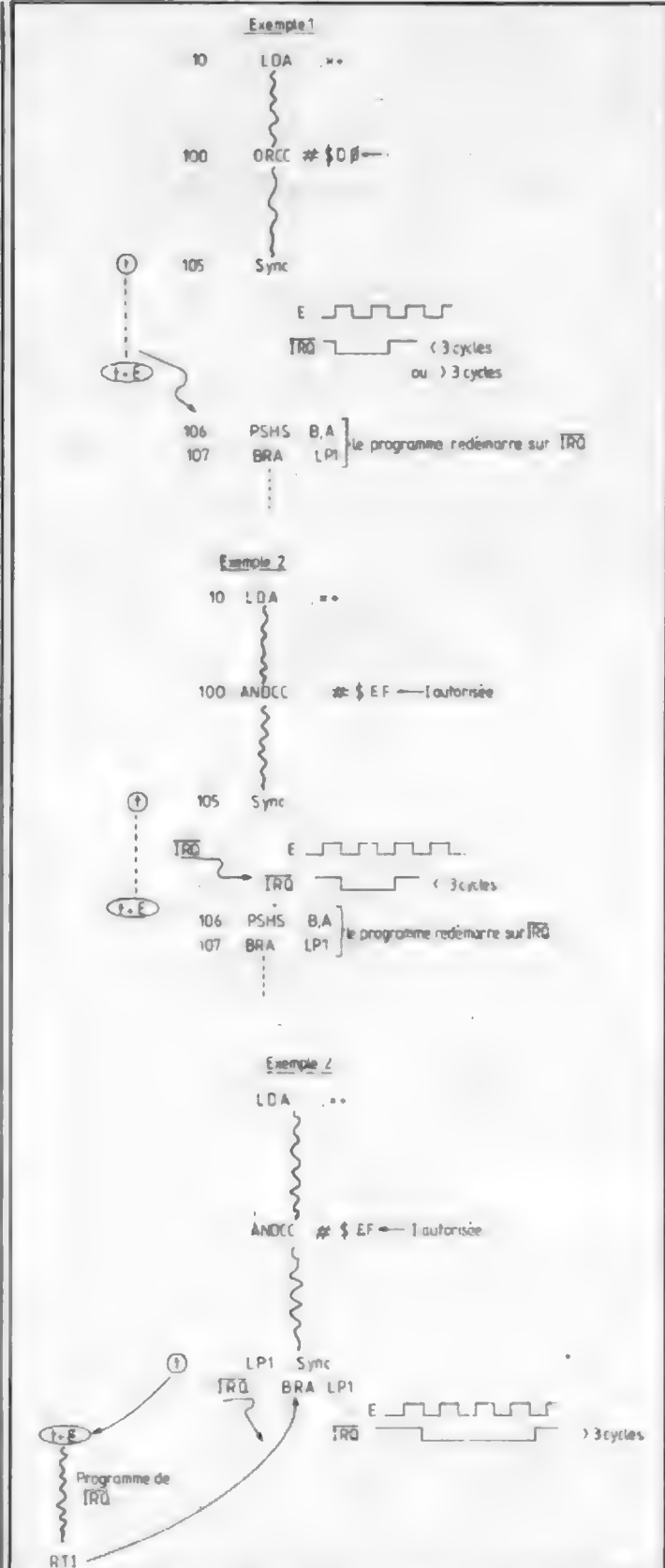
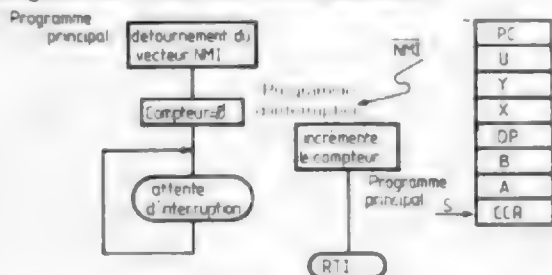


Fig. 22 : Différentes séquences d'exécution de SYNC suivant l'état de I et la durée de l'IA = 0.

### 1) Avec l'utilisation de la fonction SYNC.

Le programme principal sera logé en \$0000, tandis que celui d'interruption sera logé en \$0100.

Le détournement du vecteur NMI est simple à réaliser, celui-ci se trouve en \$07DD (voir listing du moniteur). En fait, lorsqu'une interruption NMI est reconnue par le processeur, celui-ci va chercher une adresse en \$FFFC/\$FFFD.

On y trouve \$E7EA, cette valeur est mise dans le compteur de programme et le CPU va exécuter le programme se trouvant à cette adresse, on y a :

E7EA LDX > SAVNMI B7 07DD charge dans X le contenu se trouvant à l'adresse 07DD-07DE

JMP ,X saut à l'adresse pointée par X (\$ E272)

En 07DD, on est dans la mémoire RAM, ce qui permet de lire et d'écrire (donc de détourner le vecteur qui se trouve là), il suffit donc dans le cas de notre exemple, de mettre \$0100 aux adresses \$07DD-07DE.

Le compteur peut être l'accumulateur A (ou une adresse mémoire)

Le programme sera donc le suivant :

Programme principal :	\$0000	LDX	#\$0100	BE	01	00
		STX	> SAVNMI	BF	07	DD
		CLRA		4F		
	Retour	SYNC		13		
		BRA	Retour	20	FD	
Programme d'interruption :	\$0100	INC	1,5	6C	61	
		RTI		3B		

Un appui sur la touche NMI aura pour conséquence de se dérouter sur la routine d'interruption qui se trouve en \$0100, on incrémente l'accumulateur A puis, un RTI nous ramène à l'adresse \$0008.

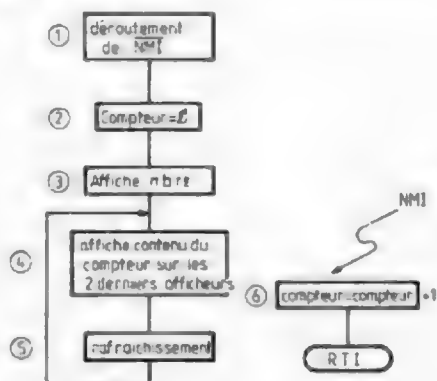
Un branchement relatif positionne le CPU en haute impédance (SYNC) pour lui permettre d'attendre une nouvelle interruption.

Un appui sur Reset, réinitialise le moniteur pour nous permettre de « reprendre la main » puis r A nous permettra de vérifier que le comptage s'est bien effectué.

### 2) Avec affichage du nombre d'appuis :

Le programme sera dans ce cas un peu plus complexe. Puisqu'il s'agit d'afficher, le CPU devra constamment travailler, donc les instructions CWAI ou SYNC seront interdites.

L'organigramme est le suivant :



Les parties 1, 2 et 6 sont identiques dans leur principe à l'exemple précédent, nous laisserons donc les instructions correspondantes en place.

La partie 3 fait appel à l'écriture de nbrE dans disbuf, on se servira de Y pour pointer sur disbuf et X contiendra la valeur à afficher : nbrE se traduit par \$45754179.

La partie 4 consiste à convertir une valeur hexadécimale se trouvant dans l'accumulateur A en une valeur d'affichage 7 segments, les sous-programmes L7SEG (\$ E0FC) et R7SEG (\$ E100) se chargeront de cette conversion (rappelons que la valeur à convertir doit obligatoirement se trouver dans A, il se trouve détérioré en fin de sous-programme ; ce qui nous amènera à utiliser B comme compteur).

La partie 5 sert à balayer les afficheurs de manière à visualiser l'affichage, le sous-programme DISPRE (\$ E07B) se chargera de cela.

D'où le programme suivant :

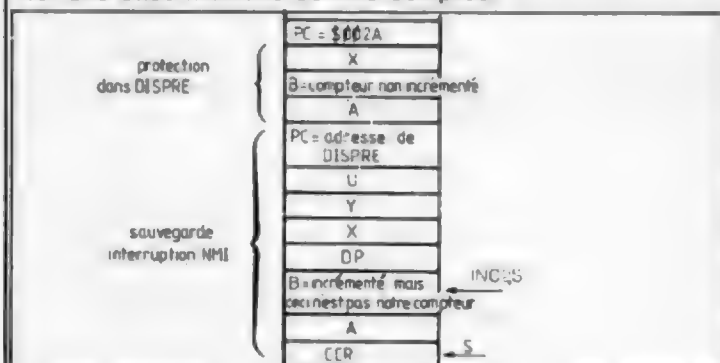
\$0000	LDX #\$0100	
	STX > SAVNMI	Détournement de NMI
	CLRB	Compteur = 0
	LDY # \$07FA	Pointe sur disbuf (1 <sup>er</sup> aff.)
	LDX#\$4575	
	STX,Y ++	Ecrit nbrE dans disbuf et pointe en disbuf +4 = 4 <sup>e</sup> afficheur
	LDX#\$4179	
	STX,Y ++	
Retour	TFR B,A	Compteur dans A
	PSHS A	Sauve compteur à convertir en 7 segments
	LBSR L7SEG	Conversion des 4 bits de poids forts
	STA,Y +	Stocke dans 5 <sup>e</sup> afficheur
	PULS A	Reprend compteur
	LSBR R7SEG	Conversion des 4 bits de poids faibles
	STA,Y	Stocke dans 6 <sup>e</sup> afficheur
	LEAY,-Y	Revient au 5 <sup>e</sup> afficheur
	LBSR DISPRE	Allume les afficheurs
002A	BRA Retour	Recommence NMI
\$0100	Inc 2,S	Incrémente B
	RTI	

Si vous essayez ce programme, vous constaterez qu'il ne fonctionne pas à tous les coups. On constate en effet que l'on effectue une sauvegarde de B (notre compteur) en Pile dans le sous-programme DISPRE.

Or, il y a de fortes chances pour que l'interruption se produise dans ce sous-programme puisque c'est lui qui réclame le plus de temps pour s'exécuter.

En conséquence, le compteur que l'on incrémente dans le programme d'interruption n'est pas notre compteur (voir dessin).

Il est donc exclu de prévoir A, B, X comme compteur, il nous reste Y et U mais la solution la plus simple étant d'utiliser une case mémoire comme compteur.



```
$0000      LDX # $0100      8E 0100
            STX > SAVNMI     BF 07DD
            CLR > CASE       7F 00 30
            LDY # $07FA      108E 07FA
            LDX # $4575       8E 4575
            STX, Y++          AF A1
            LDX # $4179       86 41 79
            STX, Y++          AF A1
```

```
$ 00 17 retour  LDA > CASE      B6 00 30
                  PSHS A        34 02

$ 001C          LBSR L7SEG      17 (E0FC) E0DD
                  STA, Y+       A7 A0
                  PULS A        35 02

$ 0023          LBR R7SEG      17 (E100) EDOA
                  STA, Y        A7 A4
                  LEAY, -Y      31 A2
                  LBSR DISPRE   17 (E07B) E04E
                  BRA retour    20 ($0017) E8

$ 0100          INC > CASE      7C 00 30
                  RTI           3B
```

Cet exemple montre bien qu'il faut être très prudent dans la gestion de la pile lors d'interruptions ; il est quelquefois préférable de se réserver une adresse buffer pour y faire un travail particulier (ici notre compteur) afin de ne pas subir une perte de données à cause d'une mauvaise gestion de la pile.

## Chapitre IV

# Aspects du Logiciel

Le logiciel présenté ici est celui du Microkit 09. Celui du MOPET, intitulé Micromon-Plus reste très semblable dans toutes ses formes et sera présenté dans le tome 2.

### Présentation Générale

Le logiciel est constitué par le programme moniteur NANO-MON REV 1.8 implanté en EPROM 2716 2 K x 8 bits depuis l'adresse E000 à l'adresse E7FF. Le programme principal s'articule autour du programme RESET comme le montre l'organigramme ci-dessous. A la mise sous tension, ou dès l'appui sur la touche «Reset» le signe «--» est visualisé sur l'afficheur de gauche. Dès lors, seules les touches M, B.P.R.CN.L.P.GO. sont influentes. L'appui sur l'une de ces touches provoque l'exécution de l'un des sous-programmes EXMEMO, BPOINT, FONREG, etc... aux adresses précisées sur l'organigramme général. Ainsi, la touche M permet l'examen et le changement du contenu des mémoires. L'emplacement mémoire visé doit être précisé par son adresse hexadécimale, entrée par l'intermédiaire du clavier et contrôlée par le sous-programme BADD. La donnée correspondante est alors affichée. Cette donnée peut être changée.

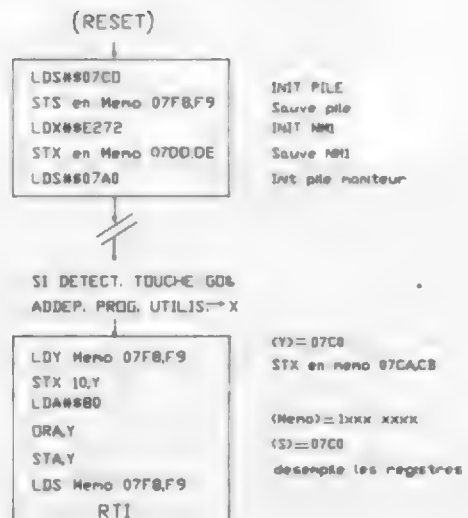
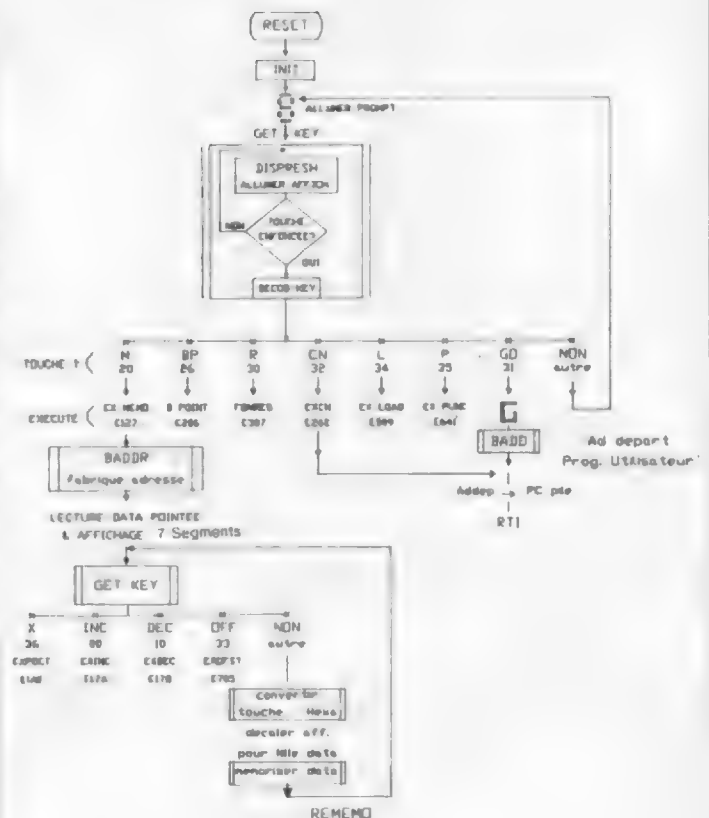
Les caractéristiques essentielles du programme moniteur sont décrites dans les paragraphes suivants. Nous analyserons les programmes et sous-programmes suivants :

- Mise en route RESET
- Clavier et affichage
- Examen et changement du contenu des mémoires
- Visu et changement du contenu des registres
- Calcul automatique d'offset
- Interface cassette

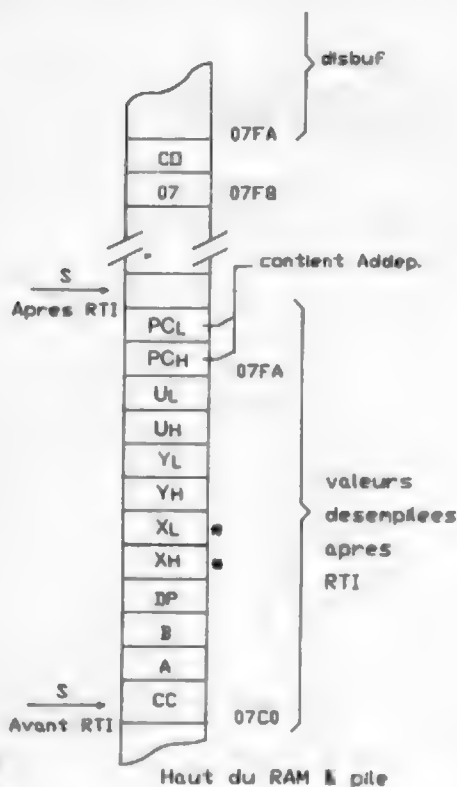
### Mise en route & Initialisations

Un appui sur la touche «RESET» force le  $\mu p$  à lire les emplacements mémoire FFFE, FFFF. Le décodage des adresses hautes ( $A_{15}, A_{14}, A_{13}$ ) du système étant partiel par tranches de 4 Koctets (1000 Hexa) et par tranches de 2 Koctets (800 Hexa) pour les adresses basses de la RAM et de l'Eprom, le  $\mu p$  prend en compte les adresses images E7FE, E7FF. Il y trouve E219 et exécute le programme «RESET» partant de cette adresse.

Ce programme initialise le pointeur de pile du système à la valeur 07C0 et le vecteur NMI à la valeur E272. Le pointeur d'interruption est mis en RAM à l'Adresse 07DD,DE. L'utilisateur peut changer cette valeur et obliger le programme d'interruption à exécuter un programme spécifique d'inter-







ruption au lieu d'exécuter celui commençant à l'adresse E272. La composition de la table des vecteurs d'interruption permet de mieux comprendre ce qui vient d'être dit.

E7F2,F3 contient 077A	RSW13		
E7F4,F5	E27C	RSW12	ASSOCIE A Break Point
E7F6,F7	0775	RFIRQ	
E7F8,F9	0770	RIRQ	
E7FA,FB	027A	RSW1	VISUREG
E7FC,FD	E7EA	RNM1	LDX Memo 07DD LJP,X
E7FE,FF	E219	RESTAR	PROGRAMME PRINCIPAL

EPROM      RAM      INTERR.      PROG.EXECUTE

C'est ensuite l'initialisation du pointeur de pile moniteur, la génération du symbole prompt « - » et le branchement au s/p GETKEY qui permet la scrutation du clavier (pour y détecter toute commande actionnée) et l'affichage. Si la touche GO a été enfoncée il faut alors entrer l'adresse de départ du programme utilisateur par l'intermédiaire du clavier contrôlé par le sous-programme BADDR (BUILD ADDRESS). Ces 2 s/p GETKEY et BADDR seront détaillés dans les chapitres suivants. Nous revenons au programme RESET avec dans X l'adresse de départ du prog. utilisateur. La valeur de X est placée dans les cases Mémoires (07CA, 07CB) qui correspondront au contenu du PC lors du désempilage provoqué par l'instruction RTI de fin de programme.

C'est ensuite la mise à 1 du bit 7 de la case Mémoire 07CC. C'est aussi la mise à 1 du flag E du registre CC vis-à-vis du désempilage. C'est enfin la réinitialisation du pointeur de pile du système et le RTI. Cette dernière instruction provoque le désempilage de tous les registres et le PC se trouve chargé à l'adressé départ prog. utilisateur. Le système exécute alors le prog. utilisateur.

**NMI** - Un appui sur la touche NMI provoque le lancement du programme d'interruption à l'adresse contenue en 07DD,DE. Si ces cases mémoires n'ont pas été forcées par l'utilisateur, elles contiennent les valeurs E272, valeur injectée par le programme RESET.

ROUNMI charge A avec le MSB de l'adresse en cours de programme (LDA 10,S ; c.-a-d. MSP du PC dans A). Si le poids fort d'adresse est  $\leq$  EQ, ROUNMI provoque un branchement à R POINT. Sous contrôle moniteur (Adresse > E000), la touche NMI a même action que la touche RESET. Si un programme utilisateur est lancé (Adresse en cours inférieure à E000 et même à 07FF dans le cas de notre système), ROUNMI provoque 1 branchement à RSWI (VISU des Registres).

### Initialisation de la fonction NMI

E272 A6	8A	A ROUNMI	LDA	10,S
0274 84	FO	A	ANDA	#\$FO
E276 81	EO	A	CMPA	#\$EO
E278 27	AD	E227	BEQ	RPOINT
E27A 20	2B	E2A7	BRA	RSWI
E7EA BE	07DD	A RNMI	LDX	SAVNMI
E7ED 6E	84	A	JMP	.X

**RSWI2** - est associé à la fonction BP (Break Point) de la façon suivante : lors de la mise d'un point d'arrêt, l'instruction utile (sauvegardée en Memo 07DF, 07FO) est remplacée automatiquement par 10 3F = instruction SWI2 (voir BPOINT). Le programme utilisateur s'arrête sur cette interruption, va lire les cases Memo E7F4,F5 pour y trouver E27C. Il exécute alors le programme RSWI2 qui consiste d'abord à recentrer le compteur programme (2 fois DEC 11,S) puis à remettre l'instruction d'origine sauvegardée en Memo 07DF,EO. Un branchement à RSWI permet de visualiser et éventuellement de changer le contenu des registres du  $\mu$ p. Après retour à R POINT ou après RESET, l'appui sur la tou-

### Initialisation de la fonction SWI2

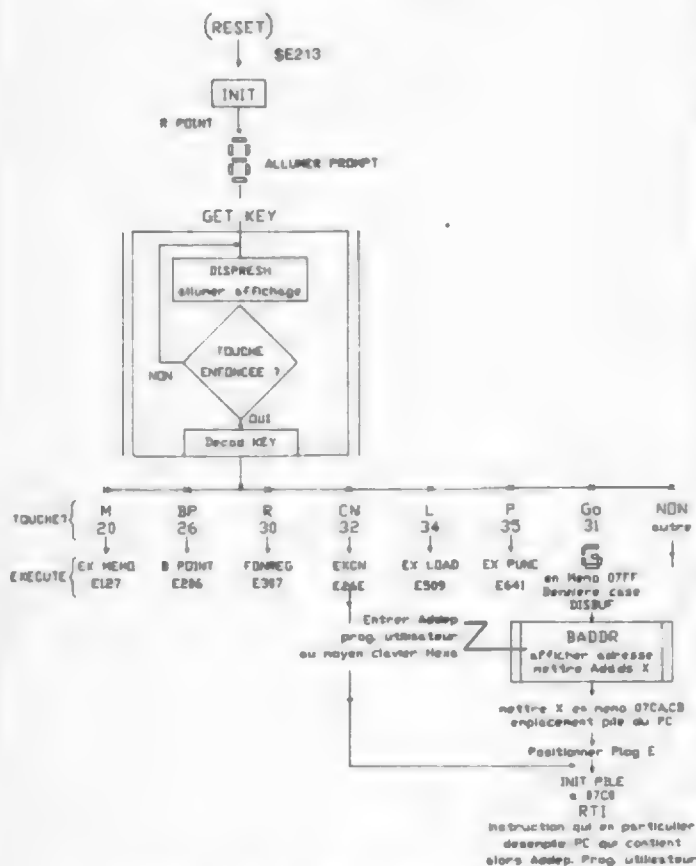
E27C 8A	6B	A RSWI2	DEC	11,S
E27E 8A	6B	A	DEC	11,S
E280 FC	07DF	A	LDD	SASWI2
E283 ED	F8 0A	A	STD	[10,S]
E286 20	1F	E2A7	BRA	RSWI

### Place un point d'arrêt, sauve l'instruction

E206 CC	756B	A BPOINT	LDD	#\$756B
E209 DD	FE	A	STD	< DISBUF + 4
E20B 17	FEA7	E0B5	LBSR	BADDR
E20E EC	84	A	LDD	.X
E210 DD	DF	A	STD	< SASWI2
E212 CC	103F	A	LDD	#\$103F
E215 ED	84	A	STD	.X
E217 20	0E	E227	BRA	RPOINT

Nous conseillons aux lecteurs de revenir à l'étude de ces deux fonctions d'interruption après s'être familiarisé avec les sous-programmes GETKEY, DISPRESH et BADDR.

Initialisation, décodage des touches M, R, CN, L, P, GO et lancement du programme utilisateur



```

E219 10CE 07ED A DESTAR LRG 0PLE DWT1 PILE
E210 10FF 07FD A STS SAMP1 ET POINTEUR E
E211 0E E212 E LRG 0000001
E224 07F 07ED A ST1 SAMP1
E227 10CE 07AO A PPOINT LRG 0P1L000
0007 A SETOP 007
E220 06 07 A LRG 0007 INIT DP
E220 1F 08 A TFR 0,0P
E227 08 57 E200 00R CL0015 DIS00W=0
E231 06 01 A LRG 0003
E235 17 FA A ST0 <DIS00W CHANGEMENT PROMPT
E235 17 F0ED E020 L0SR DETEY ALLUPE PROMPT
E230 01 26 A CWP0 0026 T0000 DP 1
E234 77 CA E204 00R 000107 001,PL0CER POINT 3'000ET
E23C 01 30 A CWP0 0030 001,EXECUTE REGISTRE1
E23E 1027 00C5 0307 LRG 0030 001,EXECUTE FONCTION DE CH04 PED
E243 01 20 A CWP0 0020 T0000=00007
E244 1027 F0FD E127 LRG E1000 001,EXECUTE ROUTINE
E240 01 33 A CWP0 0033 000, T0000<CONF100E
E246 77 22 E206 00R E000 001,EXECUTE ROUTINE
E24C 01 34 A CWP0 0034 000, T0000<L000
E250 1027 0307 E5D0 LRG E000 001,EXECUTE ROUTINE
E252 01 35 A CWP0 0035 000, T0000<PL000
E254 1027 05E9 E641 LRG E1000 001,EXECUTE ROUTINE

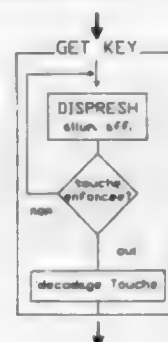
```

E250 01	SI	A	CMPO	0031	MIN,TOUCHE-0037
E250 26	CA	E237	IME	001010	MIN,DETAIIP,SCRIPTION
E250 06	7C	A	LBO	007C	001,CHANGE 6 US DOWHER DIGIT
E250 17	FF	A	S1A	<10000+3	
E260 17	FF2	E495	L000	00000	FABRIQUE ADRESSE DEPART
E260 1096	FU	A	L01	<SAMPL	DU PROGRAMME
E260 47	2A	A	S1A	10,7	ADRESSE PROG DANS PC
E260 06	00	A	L01	0000	
E260 40	00	A	000	,0	POSITIONNER PLUS E=1
E260 47	00	A	000	,7	POUR PPMOPE EN COMTE
E260 1006	FU	A	ESCH	L05	<SAMPL
E270 10			011		POUR LES REGISTRES
E270 10			011		DEPART PROG UTILISATEUR

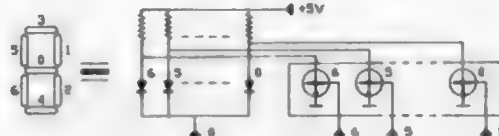
Ces deux fonctions utilisent les mêmes lignes du PIA. En outre, les deux sous-programmes :

- Affichage (DISPRESH)
- Clavier (GET KEY)

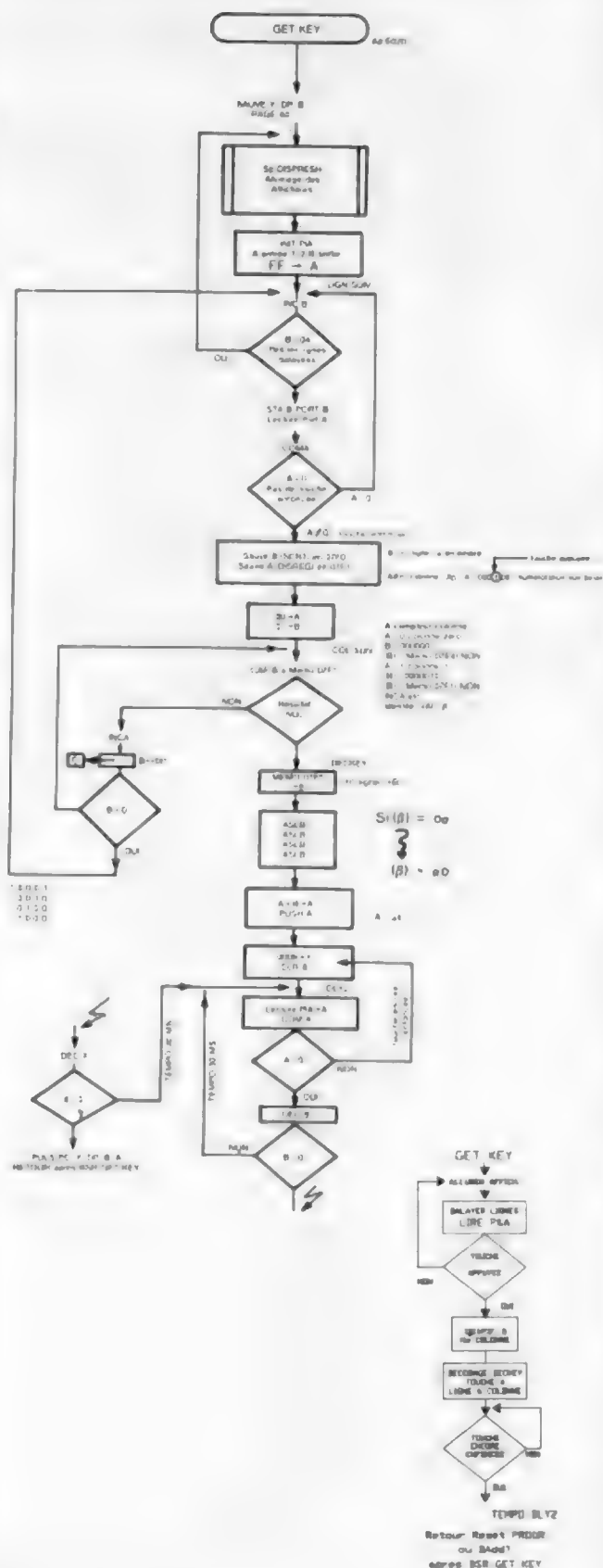
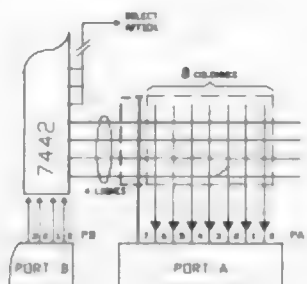
sont imbriqués ainsi que le montre l'organigramme général ci-contre. Le programme principal moniteur fait souvent appel à ces deux fonctions essentielles. Aussi nous analyserons en détail dans les paragraphes suivants les matériels et logiciels correspondants.



Il comprend 6 afficheurs à cathode commune. Pour des raisons évidentes de simplicité, les données - ou de façon plus générale - les symboles à afficher ne sont pas mémorisés mais multiplexés. Un rafraichissement (par programme) est dès lors nécessaire. Les six symboles à afficher sont (après codage 7 segments) rangés dans **6 positions mémoires** consécutives d'adresses **07FA à 07FF** (appelées **DISBUF**). Après lecture de DISBUF et aiguillage via le PIA port A, chaque bit de symbole commande l'extinction ou l'allumage d'un segment lumineux par dérivation ou non du courant d'alimentation des LEDS dans le transistor correspondant du 74LS240 (voir ci-dessous). Ceci n'est possible que si l'afficheur est sélectionné. La sélection s'effectue à partir du PIA port B (SCN REG) décodé par le 7442 qui porte la cathode de l'afficheur à zéro.





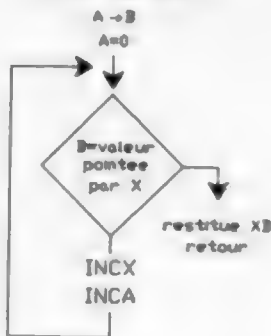






CONHEX

A = code 7 Segmt touche  
sauve X & B  
X pointe deb DIGTBL

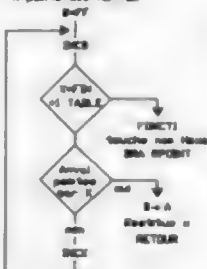


\*\*\*\*\* FABRICATION DES ADRESSES \*\*\*\*\*

E005 34	06	A	BADR	PSHS	D,A
E007 0F			CLRA		
E009 54			CLRD		
E00A F0	17FA	A	STD	DISBUF	
E00C F0	07FC	A	STD	DISBUF+2	4 PREMIERS DIGITS
E00E 0E	07EA	A	LRI	DISBUF	POINTER SUR DISBUF
E00F 00	00CC	BSP	HEINT	OFFICIE	2 PREMIERS CHIFFRES
E010 34	02	A	PSHS	A	
E012 00	04	E0CC	BSP	HEINT	OFFICIE 3e ET 4e CHIFFRE
E014 00	04	A	PULS	B	
E016 26	E3	E0AF	BRA	FABOP1	FABRIQUE ADRESSE
E018 00	16	E0E4	HEINT	BRA	VERNEE FABRIQUE VAL TOUCHE
E01C 40			ASLA		
E01D 40			ASLA		
E01E 34	02	A	PSHS	A	
E01F 00	20	E0FC	BSP	L7SEG	FABRIQUE VAL DIMENSION TOUCHE
E020 07	04	A	P7A	14	VAL CONV DANS DISBUF
E021 00	14	E0E4	BSP	HEINT	CONTINER CHIFFRES SUIVANTS
E022 00	00	A	ADDA	3	
E023 34	02	A	PSHS	A	
E024 00	20	E0FC	BSP	L7SEG	
E025 07	04	A	P7A	14	TOUCHES SUIV.
E026 00	00	A	STA	11	
E027 00	00	A	PULS	PC,4	

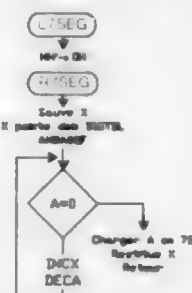
HEXCON

An de Touches B  
Sauve X  
X pointe deb DIGTBL



E02A 17	FFFA	BADR	HEINT	L7SEG	GETKEY	SCRUTER LIGNE ET COLONNES
E02B 34	14	A	HEINT	PSHS	1,8	AUTRE CHIFFRES ET 1
E02C 0E	E0D0	A	LRI	HEINT	POINTER SUR TABLEAU	DES CHIFFRES
E02D 14	FF	A	LRI	HEINT	POINTER SUR TABLEAU	DES CHIFFRES
E02E 3C			SCUTE	HEINT		
E02F 0E	E0D0	A	HEINT	HEINT	HEINT	HEINT
E030 27	24	E11E	BED	FUNCT1	001, C'EST UNE FONCTION	
E031 00	00	A	CHPA	001	NON, CHIFFRE TROUVE	
E032 26	F4	E0EE	BNE	SCUTE	NON, CONTINER A SUIVANTS	
E033 1F	70	A	TFR	D,A		
E034 35	94	A	PULS	PC,1,8	QUE, AVAL TOUCHE	

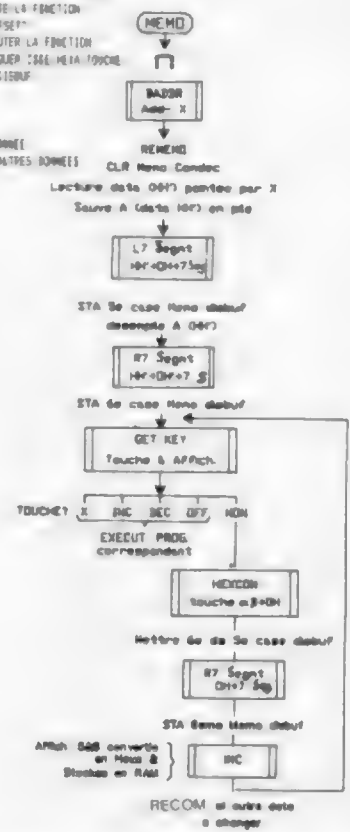
E03C 07			L7SEG	BADR	
E03D 07			BADR		
E03E 07			ASPA		
E03F 07			ASPA		
E100 34	10	A	R7SEG	PSHS	1
E101 0E	E014	A	LRI	HEINT	POINTER SUR TABLEAU
E102 04	0F	A	HEINT	POINTER SUR TABLEAU	PREMIERE TOUCHE
E103 27	05	E10E	HEINT	HEINT	HEINT
E104 30	01	A	LEAT	1,7	NON, POINTER SUR VAL SUIVANTE
E105 44			DECA		
E106 00	F0	E107	BRA	HEINT	HEINT
E107 04	04	A	HEINT	HEINT	HEINT
E108 75	40	A	PULS	PC,1	DANS A TROUVEE



E007 1E	00	A	FABOP1	E15	A,B
E007 1F	00	A	TFR	D,A	
E007 20	00	A	PULS	PC,D,A	

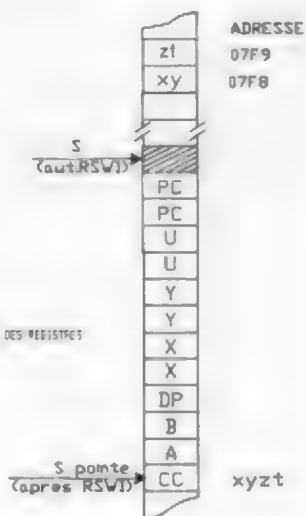
\*\*\*\*\* EXECUTION DE LA FONCTION MEMO \*\*\*\*\*

E107 0F			HEINT	CLRA	
E108 07	FE	A	STA	DISBUF+4	ETATROUPE Sur DISBUF
E109 04	0E	A	LRI	PSHS	
E10A 07	AF	A	STA	DISBUF+5	9 TAMS de DIGIT
E10B 00	01	E10C	BRA	BADR	FABRIQUE ADRESSE DANS 1
E10C 06	0E	A	HEINT	CLRA	CLRA
E10D 04	04	A	LRI	PSHS	1
E10E 34	02	A	PSHS	A	
E10F 00	C4	E0FC	BSP	L7SEG	FABRIQUE CODE A MEMO
E110 07	FE	A	STA	DISBUF+4	DANS LE "de DISBUF"
E111 35	02	A	PULS	A	
E112 00	C0	E100	BRA	P70E0	FABRIQUE CODE A MEMO
E113 07	FF	A	STA	DISBUF+5	DANS LE "de DISBUF"
E114 17	FE0	E12A	HEINT	DISBUF	ALLUMER LES DIGITS
E115 01	74	A	CHPA	000	TOUCHEMENT
E116 27	67	E10E	BED	EXPDET	001, SCUTER TOUCHE OFFSET
E117 01	00	A	CHPA	000	INCREMENTE CASE MEMO
E118 27	2F	E170	BED	EXPDET	001, LITCUTER LA FONCTION
E119 01	01	A	CHPA	001	NON, DECREMENTE CASE MEMO
E120 27	29	E170	BED	EXPDET	001, EXECUTER LA FONCTION
E121 01	02	A	CHPA	002	TOUCHE OFFSET
E122 1027	00D0	E170	LRI	EXPDET	001, EXECUTER LA FONCTION
E123 00	70	E0E7	BSP	HEINT	NON, FABRIQUE CODE MEMO TOUCHE
E124 04	FF	A	LRI	DISBUF+5	SHIFTER DISBUF
E125 07	FE	A	STA	DISBUF+4	
E126 00	00	E100	BRA	P70E0	
E127 07	FF	A	STA	DISBUF+5	
E128 00	02	E165	BRA	INCHEN	STOCKER DONNEE
E129 20	00	E140	BRA	HEINT	RECON SI AUTRES DONNEES

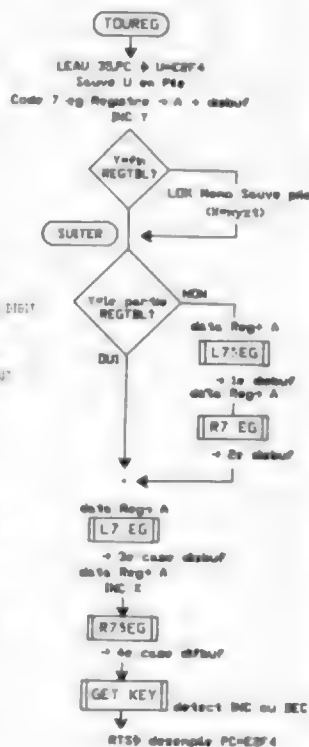




0007 A SETP 001  
 E207 00 02 A PSMT 100 0007  
 E208 1F 00 A TPR A DP  
 E209 100F 10 A STB 100001  
 E210 100C 1704 0 LPS 011000  
 E211 00 00 E200 000 CURDIS TOUT DISPARAÎT  
 E212 00 02 A LPS 0000 COMPTER REGISTRES  
 E213 00 00 A LPS 100001 TOUT CHARGEMENT DES REGISTRES  
 E214 01 00 E21 LPS REGTL, PCY + REGTL  
 LEAT REGTL, PCY + REGTL

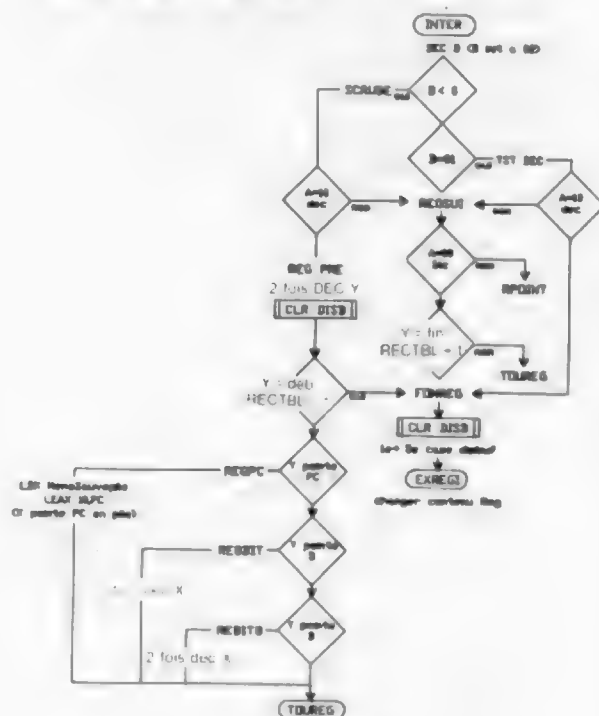


E200 00 00 0000 TOUTES LEAT INTER, PCY  
 E201 04 00 A PSMT 100 0007  
 E202 00 00 A LPS 100001  
 E203 00 00 A STA 100001  
 E204 00 00 A LPS 100001  
 E205 00 00 A LPS 100001  
 E206 00 00 A LPS 100001  
 E207 00 00 A LPS 100001  
 E208 00 00 A LPS 100001  
 E209 00 00 A LPS 100001  
 E210 00 00 A LPS 100001  
 E211 00 00 A LPS 100001  
 E212 00 00 A LPS 100001  
 E213 00 00 A LPS 100001  
 E214 00 00 A LPS 100001  
 E215 00 00 A LPS 100001  
 E216 00 00 A LPS 100001  
 E217 00 00 A LPS 100001  
 E218 00 00 A LPS 100001  
 E219 00 00 A LPS 100001  
 E220 00 00 A LPS 100001



E204 00 00 0000 INTER DEC  
 E205 00 00 0000 DEC  
 E206 00 00 0000 DEC  
 E207 00 00 0000 DEC  
 E208 00 00 0000 DEC  
 E209 00 00 0000 DEC  
 E210 00 00 0000 DEC  
 E211 00 00 0000 DEC  
 E212 00 00 0000 DEC  
 E213 00 00 0000 DEC  
 E214 00 00 0000 DEC  
 E215 00 00 0000 DEC  
 E216 00 00 0000 DEC  
 E217 00 00 0000 DEC  
 E218 00 00 0000 DEC  
 E219 00 00 0000 DEC  
 E220 00 00 0000 DEC

E210 00 00 0000 DEC  
 E211 00 00 0000 DEC  
 E212 00 00 0000 DEC  
 E213 00 00 0000 DEC  
 E214 00 00 0000 DEC  
 E215 00 00 0000 DEC  
 E216 00 00 0000 DEC  
 E217 00 00 0000 DEC  
 E218 00 00 0000 DEC  
 E219 00 00 0000 DEC  
 E220 00 00 0000 DEC

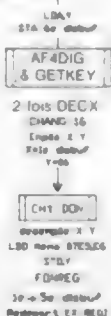


00000 PERRE DE CHANGER LE CONTENU S IN REG LORS S UN NRI DE S01 00000

E204 00 00 0000 INTER DEC  
 E205 00 00 0000 DEC  
 E206 00 00 0000 DEC  
 E207 00 00 0000 DEC  
 E208 00 00 0000 DEC  
 E209 00 00 0000 DEC  
 E210 00 00 0000 DEC  
 E211 00 00 0000 DEC  
 E212 00 00 0000 DEC  
 E213 00 00 0000 DEC  
 E214 00 00 0000 DEC  
 E215 00 00 0000 DEC  
 E216 00 00 0000 DEC  
 E217 00 00 0000 DEC  
 E218 00 00 0000 DEC  
 E219 00 00 0000 DEC  
 E220 00 00 0000 DEC

EX00 34	30	A	EMMIA	PS05	1,1
EX00 0E	37FA	6		LB7	01030UF
EX00 100E	0000	A		LB7	0000
EX01 17	2000	EQRE		LBSP	CH0000
EX04 72	20	A		PA05	1,1
EX06 0C	ES	6		LB0	00000000
EX00 0B	04	A		ST0	1
EX04 16	0020	ES07		LB00	000000

1 point: Das Reg on side  
LPR memo 578

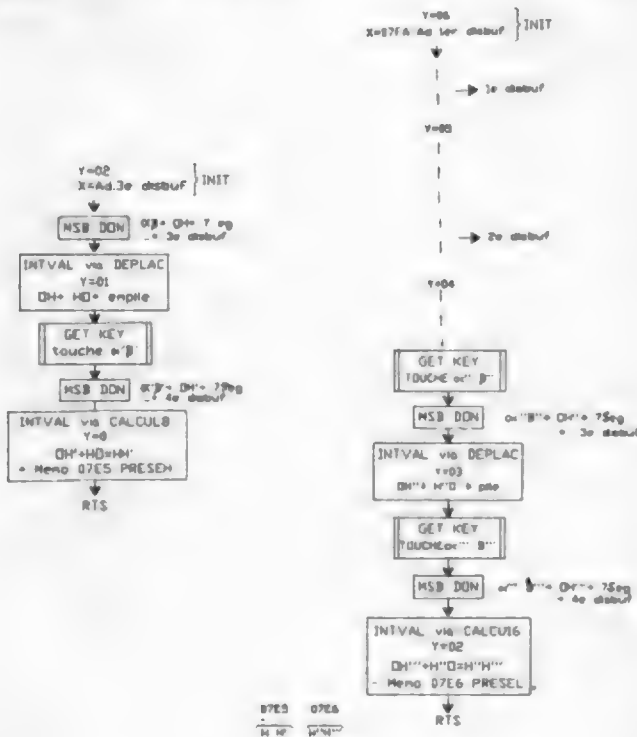


E4E2 35	92	A	DEPLAC	PULS	6
E4E4 48				ASLA	
E4E5 48				ASLA	
E4E6 48				ASLA	
E4E7 48				ASLA	
E4E8 34	02	A		PMS	8
E4E9 20	0C	E4E9		DWA	OUTEP
E4E5 35	02	A	CALCUL	PLAS	8
E4E6 48	09	A		ADDA	, 54
E4F0 47	ES	A		STA	-PRESSEN
E4F2 29	0F	E4E5		DWA	OUT



Les figures ci-contre représentent l'exécution de CHTDON selon l'initialisation préalable de X et de Y (Ad E385 et Ad E3CA de BX REGI) :

a) Chargement de la valeur hexadécimale HH' dans la memo PRESEH 07E5 et transcodage en 7 segments pour afficher cette valeur sur les 3<sup>e</sup> et 4<sup>e</sup> afficheurs. Ces valeurs hexa sont entrées par le clavier sous contrôle de GETKEY.  
b) Même chose qu'en a) pour une valeur hexadécimale HH' H'' H'''. Dans ce dernier cas H'' H''' sont stockés en Memo PRESEL 07E6.



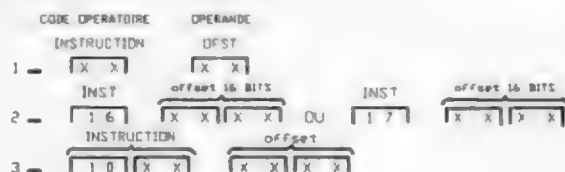
## CALCUL AUTOMATIQUE D'OFFSET

Le logiciel NANOMON permet le calcul automatique dans les deux cas suivants :

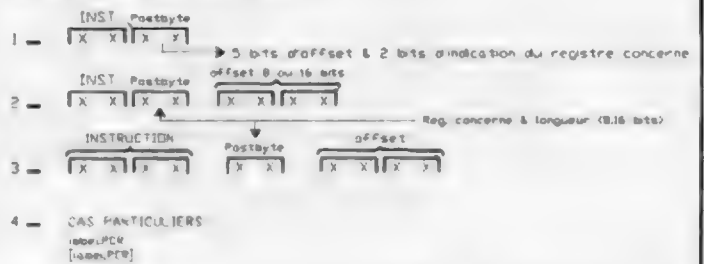
- en adressage relatif (prog.EXOFS)
- en adressage indexé (prog.EXPOCT)

Nous analyserons successivement les deux programmes EXOFS et EXPOCT. Nous proposons cependant, au préalable, 2 tableaux en guise de rappels.

### OFFSET EN ADRESSAGE RELATIF (Branchement)



### OFFSET EN ADRESSAGE INDEXÉ



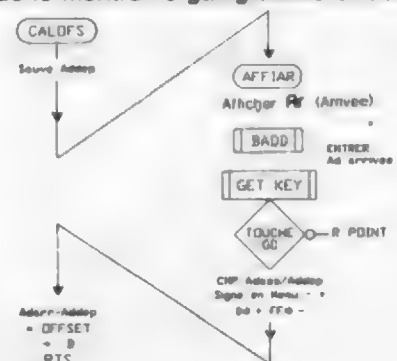
RR = 00 = X  
RR = 01 = Y  
RR = 10 = U  
RR = 11 = S  
I = 1 = Indirect  
I = 0 = Direct  
# = 0 calculer  
X = Indifferent

BINAIRE	HEXA	SIGNIFICATION
1RR00000	#0	INC +
1RR10001	#1	INC ++
1RR00010	#2	DEC -
1RR10011	#3	DEC --
1RR10100	#4	offset nul
1RR10101	#5	offset AccB
1RR10110	#6	offset AccA
1RR11000	#8	offset 8 bits
1RR11001	#9	offset 16 bits
1RR11011	#B	offset AccD
1XX11100	#C	offset PCRB bits
1XX11101	#D	offset PCRL6 bits
1XX11111	#F	Indirect attendu

## EXOFST (Ad E705)

### Calcul de l'Offset en cas de branchement

Ainsi que le montre l'organigramme de la page suivante, le programme EXOFST, vérifie d'abord le dernier octet entré en RAM (et éventuellement l'avant dernier octet) pour déterminer si l'offset de branchement doit comporter 8 ou 16 bits (branchement court ou branchement long). L'exécution d'OFST se poursuit ensuite par le calcul proprement dit de l'offset par appel au sous-programme CALOFST, qui calcule la différence entre l'adresse d'arrivée et l'adresse de départ, ainsi que le montre l'organigramme ci-dessous :





00000 CALCULE L'OFFSET SUR 16 BITS 00000  
00000 CONTIENT L'OFFSET, L'ADRESSE DEP 000

E757 1F 12	A	CALOPS 1F0	1,7	ADRESSE DE DEPART
E759 1944 EA	A	STY	00000	
E75C 00 05	E763	050	AFFIAR	AFFICHE ADRESSE ARRIVEE
E75E 1F 10	A	1F0	1,7	D'ADRESSE DEPART
E760 95 EA	A	0000	00000	D'ADRESSE DEPART+OFFSET
E762 39		RTS		

00000 AFFICHAGE DE L'ADRESSE D'ARRIVEE 00000  
00 STOCKER DES NEGROS LE SENS DE L'OFFSET + 000

E762 CC	0F41	A	AFFIAR LDB	00AF41	AFFICHE ADRESSE DISBUR
E764 00	FE	A	STD	-DISBUR+4	
E766 17	FA0A E005	LDBR	0A000	AFFICHE ADRESSE D'ARRIVEE	
E768 0F	EC	A	STY	000000	
E76A 17	FA0A E020	LDBR	0E000	DEFEV	
E76C 01	71	A	CPA	0A21	"BUENE 00"
E76E 27	03	E777	0E0	SUIVIT	OUT, CHARGE ADRESSE
E770 1A	FA0A E027	LDBR	00000	00000	
E772 17	FA0A E200	SUIVIT	LDBR	CL0015	ETENDRE AFFICHAGE
E774 06	FF	A	LDA	0001	
E776 1C	EA	A	CPA	AD000	ARRIVEE DEPART
E778 25	01	E781	0A0	STACAA	OUT, STOCKER DES NEGROS
E780 AC			INCX		
E781 97	EY	A	STACAA STA	000000	NEGROS NEGROS+10 NEGROS+FF 0
E783 39		RTS			

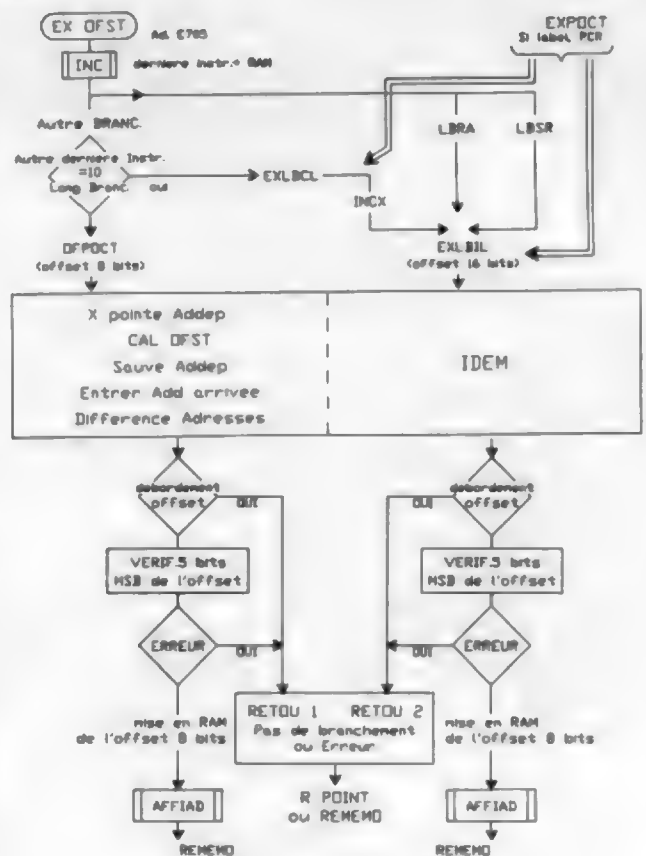
Au retour de CALOFST, la Memo 07E9 contient le sens de déplacement et l'accumulateur D l'offset sur 16 bits. Le programme principal se poursuit par des tests sur le résultat de l'offset calculé notamment vérification de non dépassement de capacité, et vérification de la longueur (8 bits, 16 bits) du déplacement... Le symbole Er est affiché en cas d'erreur. Le programme se termine par le rangement de l'offset dans le prog. utilisateur et par l'affichage de cet offset dans la nouvelle adresse de départ.

00000 AFFICHAGE DE L'ADRESSE DE CHARGEMENT 00000  
00 DE L'OFFSET DANS LES QUATRE PREMIERS S1GITS 000

TFRY → X → D → disbur

E7ED 1F 21	A	AFFIAR 1F0	1,3	
E7ED 1F 10	A	TFR	1,0	NLE ADRESSE DE DEPL. APRES INSTRUCT
E7EF 17	FA0A E190	LDBR	01900	
E7F0 01	FD	A	STA	-DISBUR+1
E7F2 1F 01	A	TFR	1,0	
E7F4 17	FA0A E19C	LDBR	17900	
E7F6 01	FA	A	STA	-DISBUR
E7F8 1F 00	A	TFR	0,A	
E7FA 17	FA0A E190	LDBR	01900	
E7FC 01	FD	A	STA	-DISBUR+3
E7FE 1F 00	A	TFR	0,A	
E7F0 17	FA0A E19C	LDBR	17900	
E7F2 01	FE	A	STA	-DISBUR+2
E7F4 39		RTS		

## Organigramme général Offset de branchement



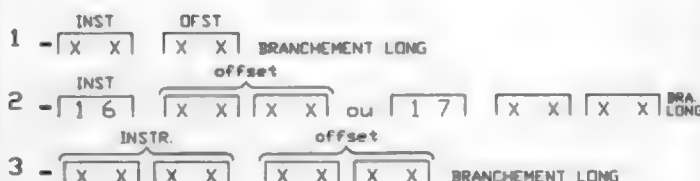
## Exécution de la fonction Offset

### Organigramme Exofst

EXOSFT vérifie l'instruction de branchement puis calcule l'offset en fonction de l'adresse d'arrivée désirée, laquelle est entrée par l'utilisateur par l'intermédiaire du clavier hexa. L'offset ne sera placé en RAM (dans le programme utilisateur) que si les vérifications qu'il subit s'avèrent positives.

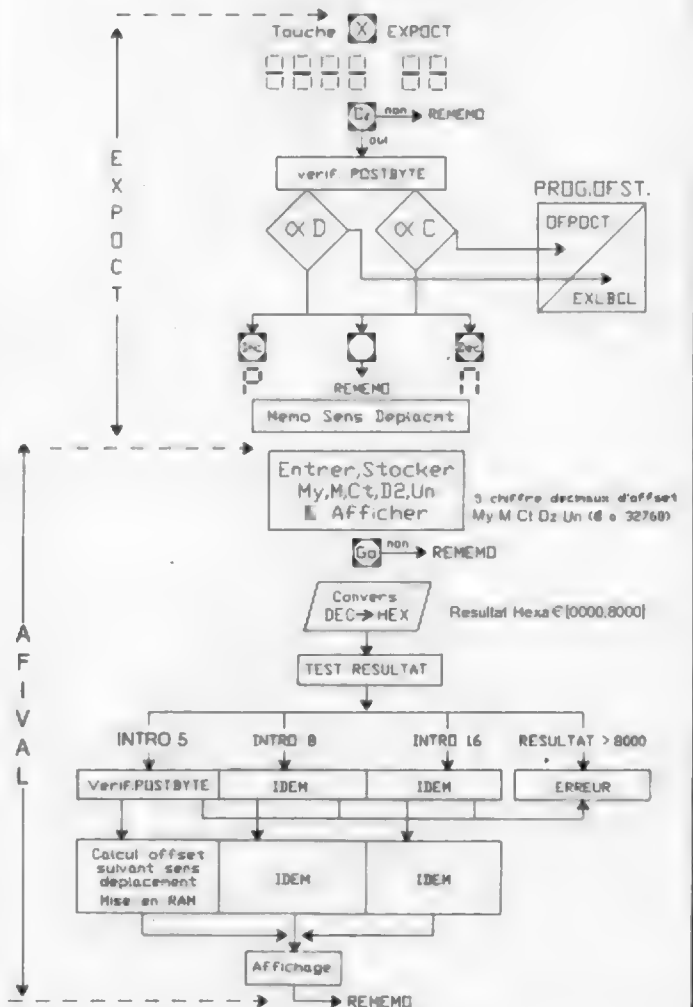
E705 17	FA0A E163	E30FST LDBR	INCXEN		
E706 A6	04	A	LDA	1,7	
E708 01	16	A	CPA	0116	LDBR
E70C 27	34	E742	0E0	EXLDBL	OUT, BRANCHEMENT LONG INCONDITIONNEL
E70E 01	17	A	CPA	0117	LDBR
E710 27	30	E742	0E0	EXLDBL	OUT, BRANCHEMENT LONG INCONDITIONNEL
E712 06	02	A	LDA	1,0	POINTER SUR INSTRUCTION PRECEDENTE
E714 01	10	A	CPA	0110	BRANCHEMENT CONDITIONNEL LONG
E716 27	20	E740	0E0	EXLDBL	OUT
E718 30	03	A	OPFCT	LEAD	1 CONTIENT ADRESSE DU BRANCH COURT
E71A 00	30	E757	DOP	CALOPS	CALCULER OFFSET
E71C 29	72	E790	DVS	RETOUT	SI DEPASSEMENT DE CAPACITE, ERREUR
E71E 95	E9	A	DITA	REPOS	
E720 27	04	E726	0E0	BRAMO	TOUT MED A 0 = BRANCH POS

### Offset en cas de branchement



[illegible]

E704 39		PLCOUT	PLD3		
E705 25	00	E704	0C5	RETOUR	BRANCHE COURT POSSIBLE
E707 26			REPOB		
E708 20	06	E708	004	SAUTER	CONTINUE CALCUL
E706 39			DECOPI1		
E700 24	07	E700	0CC	RETOUR	
E700 56			REPOB		
E706 20	CO	E706	004	SAUTER	
E707 31	3E	A RETOUR	LEAF	-2,7	
E702 29	10	E701	00F	CLIGND	APPRICH EPREUVE, PLUS RETOUR MEMOIR
E706 31	3E	A RETOUR	LEAF	-4,7	
E706 06	00	A	L0A	-7,7	
E700 01	10	A	CPMA	0010	
E706 27	17	E70F	000	CLIGND	
E70C 06	04	A	B0A	-7	
E706 01	16	A	CPMA	0010	
E704 27	0F	E701	000	CLIGND	
E702 01	17	A	CPMA	0017	
E704 27	00	E701	000	CLIGND	
E706 04	0F	A	CPMA	000F	
E700 01	00	A	CPMA	0000	
E704 27	05	E701	000	CLIGND	
E70C 16	00	E70C E227	LB06	PP0100	SI P06 BRANCHE, RETOUR RESE?
E70F 21	3F	A CLIGND	LEAF	-2,7	
E701 03	10	E70C	CLIGND	PP01	APPRICH ADRESSE DU EST L'ENFER
E705 03	09	E70E	000	0000	PLAC L'ENFER 05 DISJON
E705 100E	00FF	A	L07	002F7	
E70F 17	00F	E70B	AFCL10	LB00	DISPPE
E70C 20	06	E70C	00A	TEMP	APPRICH L'ENNEUR
E70C 00	7F01	A	ERROR	L00	TEMPORISATION
E701 00	FE	A	57D	007F01	PLACE 0r 2005 LES 2
E70C 29			RTS	-DISJON-4	DEPHIEND 010175
E704 31	3F	A TEMPO	LEAF	-1,7	
E70C 26	F1	E70D	00F0E		
E70C 16	F045	E70D	000A	00000	RETOUR 05 PC7 MEMORY



## Organigramme début EXPOCT

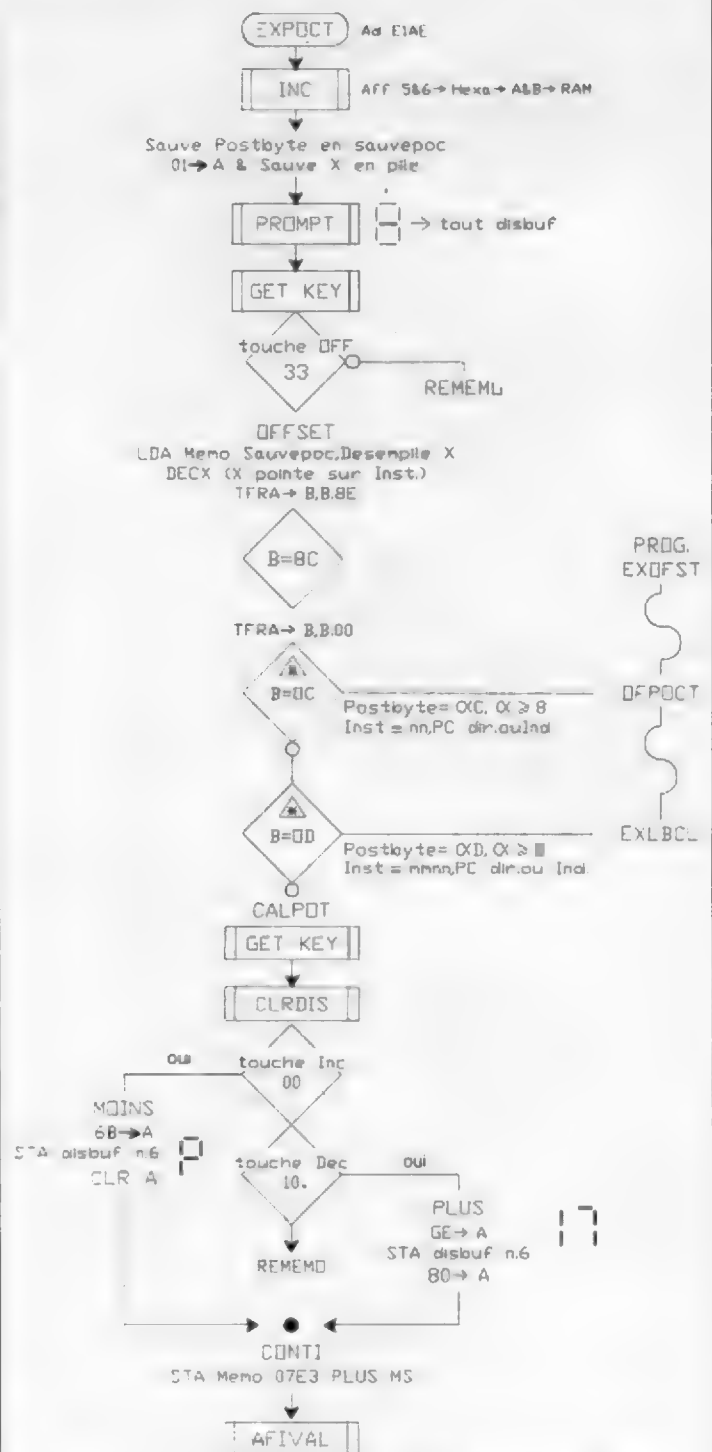
La première partie d'EXPOCT contrôle les touches **X**, **OP**, puis **AC** ou **PC**.

Ce début de programme vérifie aussi le postbyte et remplit la mémoire PLUSMS en fonction du sens souhaité du déplacement. La suite du programme est appelée AFIVAL.

E1A0 00	00	E1A5 EXPOCT 050	INCDEC	POST OCTET EN MEM
E1A1 01	01	A STA	SAUVE	SAUVE POST OCTET
E1A2 06	01	A LDA	0041	ALLURE PROMPT
E1A3 24	10	A PSHS	1	
E1A4 17	0040 E291	LDSH	PROMPT	
E1A5 17	F6A4 E029	LDSH	GETKEY	
E1A6 01	20	A CHPA	0023	TOUCHE OFFSET
E1A7 27	03 E1C3	DEB	OFFSET	
E1A8 16	F7A0 E130	LDBA	REMEM	NON, RETIENS PC EN MEM
E1A9 06	04	A OFFSET	LEA	SAUVE
E1B0 25	10	A	PULS	1
E1B1 30	10	A	LEAD	-1,2
E1B2 1F	09	A	TFB	A, B
E1B3 0E	0E	A	AND	000E
E1B4 01	0C	A	CHPB	000C
E1B5 26	10	E1B1	ONE	CALPOT
E1B6 1F	09	A	TFB	A, B
E1B7 04	00	A	AND	0000
E1B8 01	0C	A	CHPB	000C
E1B9 1A27	0510 E130	LDBD	EXPOCT	
E1BA 01	00	A	CHPB	0000
E1BB 1A27	0510 E130	LDBD	EXPOCT	
E1BC 17	F6A4 E029	LDSH	GETKEY	
E1BD 00	00	A	CHPA	0000
E1BE 27	04 E1B4	DEB	PLUS	POSTOCTET 0
E1BF 00	00	A	CHPA	0000
E1C0 27	0C E1B0	DEB	PLUS	POSTOCTET 0
E1C1 30	00	A	CHPB	0000
E1C2 16	F7A0 E130	LDBA	REMEM	NON, RETIENS PC EN MEM
E1C3 06	04	A PLUS	LEA	0004
E1C4 01	0C	A	CHPB	000C
E1C5 0F	00	A	CHPB	000F
E1C6 20	06 E201	LDBA	CONT	
E1C7 06	06	A PLUS	LEA	0006
E1C8 0F	00	A	CHPB	000F
E1C9 06	00	A	CHPB	0006
E1CA 0F	00	A	CHPB	000F
E1CB 0F	00	A	CHPB	000F
E1CC 0F	00	A	CHPB	000F
E1CD 0F	00	A	CHPB	000F
E1CE 0F	00	A	CHPB	000F
E1CF 0F	00	A	CHPB	000F
E1D0 0F	00	A	CHPB	000F
E1D1 0F	00	A	CHPB	000F
E1D2 0F	00	A	CHPB	000F
E1D3 0F	00	A	CHPB	000F
E1D4 0F	00	A	CHPB	000F
E1D5 0F	00	A	CHPB	000F
E1D6 0F	00	A	CHPB	000F
E1D7 0F	00	A	CHPB	000F
E1D8 0F	00	A	CHPB	000F
E1D9 0F	00	A	CHPB	000F
E1DA 0F	00	A	CHPB	000F
E1DB 0F	00	A	CHPB	000F
E1DC 0F	00	A	CHPB	000F
E1DD 0F	00	A	CHPB	000F
E1DE 0F	00	A	CHPB	000F
E1DF 0F	00	A	CHPB	000F
E1E0 0F	00	A	CHPB	000F
E1E1 0F	00	A	CHPB	000F
E1E2 0F	00	A	CHPB	000F
E1E3 0F	00	A	CHPB	000F
E1E4 0F	00	A	CHPB	000F
E1E5 0F	00	A	CHPB	000F
E1E6 0F	00	A	CHPB	000F
E1E7 0F	00	A	CHPB	000F
E1E8 0F	00	A	CHPB	000F
E1E9 0F	00	A	CHPB	000F
E1EA 0F	00	A	CHPB	000F
E1EB 0F	00	A	CHPB	000F
E1EC 0F	00	A	CHPB	000F
E1ED 0F	00	A	CHPB	000F
E1EE 0F	00	A	CHPB	000F
E1EF 0F	00	A	CHPB	000F
E1F0 0F	00	A	CHPB	000F
E1F1 0F	00	A	CHPB	000F
E1F2 0F	00	A	CHPB	000F
E1F3 0F	00	A	CHPB	000F
E1F4 0F	00	A	CHPB	000F
E1F5 0F	00	A	CHPB	000F
E1F6 0F	00	A	CHPB	000F
E1F7 0F	00	A	CHPB	000F
E1F8 0F	00	A	CHPB	000F
E1F9 0F	00	A	CHPB	000F
E1FA 0F	00	A	CHPB	000F
E1FB 0F	00	A	CHPB	000F
E1FC 0F	00	A	CHPB	000F
E1FD 0F	00	A	CHPB	000F
E1FE 0F	00	A	CHPB	000F
E1FF 0F	00	A	CHPB	000F

Le tableau ci-dessous rappelle la signification du postbyte.

POSTBYTE	SIGNIFICATION
0 R R * * * *	offset 5 bits
1 R R 0 0 0 0	* 0 INC +
1 R R 1 0 0 0	* 1 INC + +
1 R R 0 0 0 1	* 2 DEC -
1 R R 1 0 0 1	* 3 DEC - -
1 R R 1 0 1 0	* 4 offset nul
1 R R 1 0 1 1	* 5 offset AccB
1 R R 1 0 1 1	* 6 offset AccA
1 R R 1 1 0 0	* 8 offset 8 bits
1 R R 1 1 0 1	* 9 offset 16 bits
1 R R 1 1 0 1	* B offset AccD
1 X X 1 1 1 0	* C offset PCR, 8 bits
1 X X 1 1 1 0	* D offset PCR, 16 bits
1 X X 1 1 1 1	* F Indirect Étendu
BINAIRE	HEXA

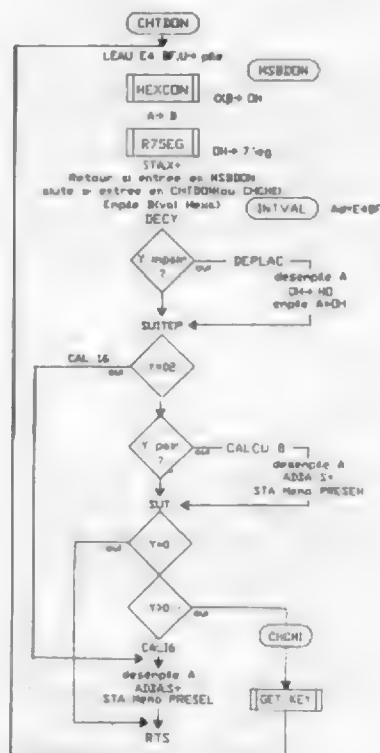


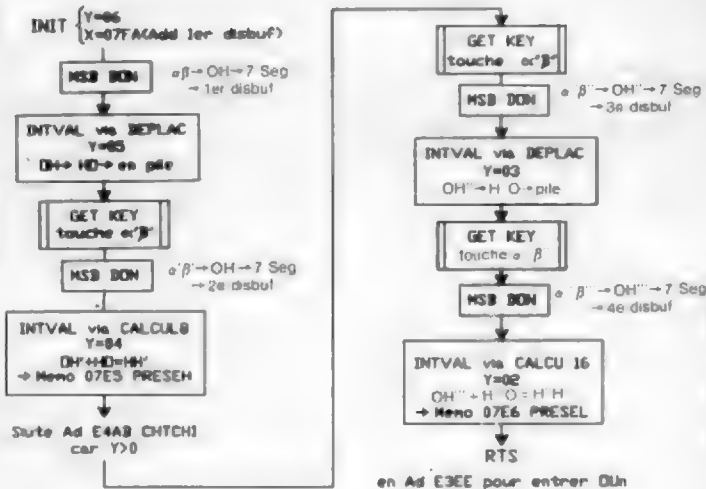
## Organigramme suite EXPOCT début AFIVAL

L'appel en cascade des s/p GETKEY et CHTDON (initialisation X = 1<sup>er</sup> disbuf Y = 06) permet de rentrer 4 chiffres à partir du clavier et de les classer en Memo 07E5, 07E6 (cf. chapitre RSWI). Un nouvel appel GETKEY/MSBDON permet de rentrer un 5<sup>e</sup> chiffre. Ces 5 chiffres sont convertis en hexa. Le résultat est testé afin de mesurer la longueur de l'offset (5, 8, 16 bits).

```

graph TD
    Start([AFIVAL E3DD]) --> IncX[INC X (X pointe postbyte)]
    IncX --> GetKey1[GET KEY]
    GetKey1 --> SetX[X=1er disbuf Y=06]
    SetX --> CHTD0H[CHT D0H]
    CHTD0H --> GetKey2[GET KEY]
    GetKey2 --> MSBDDN[MSBDDN]
    MSBDDN --> CLRP[CLR PRELVD STB PRELVD+1 Desemplé X]
    CLRP --> Touche{touche}
    Touche --> REMEMO
    REMEMO --> CALCOM[CALCOM]
    CALCOM --> CLRDIS[CLRDIS]
    CLRDIS --> Calc[DEC -> HEXA  
Un  
DA x Dz  
64 x Ct  
36B x H  
2710 x My  
Σ -> Memo 07E7.B  
My > 3 -> ERREUR]
    Calc --> LDD[LD PRELVD @MEMO 07E7?]
    LDD --> A0{A=0}
    A0 --> AB{A < 80 B > 80}
    AB --> Intro16[INTRO 16]
    A0 --> B80{B=80}
    B80 --> IntroB1[INTRO B]
    A0 --> B7F{B < 7F}
    B7F --> B40{B > 40}
    B40 --> IntroB2[INTRO B]
    B7F --> B3F{B < 3F}
    B3F --> B20{B > 20}
    B20 --> IntroB2
    B3F --> B1F{B < 1F}
    B1F --> Intro5[INTRO 5]
  
```

[illegible]



## Organigramme suite EXPOCT, fin AFIVAL - Partie INTRO5

Les cinq chiffres décimaux du déplacement entrés à partir du clavier ont été convertis en Hexa. Le résultat Hexa subit différents tests afin de mesurer la longueur (5, 11 ou 16 bits) de l'offset. En particulier, un déplacement compris entre  $[0, +15]_{dec}$ , c'est-à-dire  $[0, 0F]_{hexa}$  est traduit par un offset codé sur 5 bits et ce quel que soit le sens de déplacement.

Si le déplacement est positif, l'offset sera égal au déplacement transcodé hexa alors que si le déplacement est négatif, l'offset sera le complément à 2 du nombre hexa. Notons encore qu'à 1 déplacement négatif de 16 correspond aussi 1 offset codé sur 5 bits.

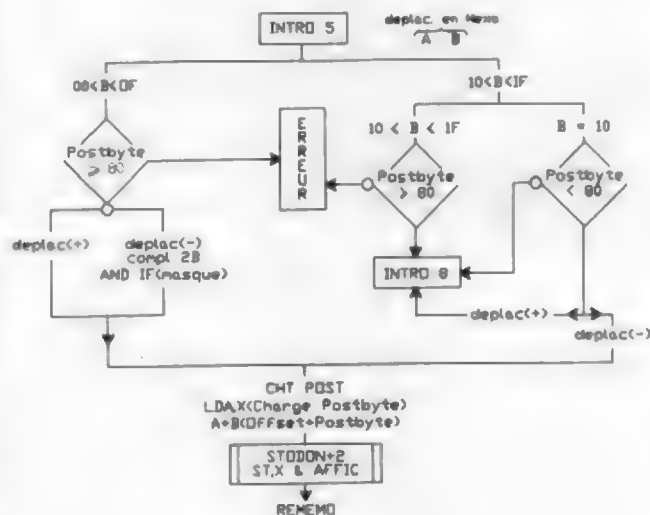
Dans le cas du codage sur 5 bits, l'offset devant être inclus dans le postbyte, l'utilisateur devra « écrire » un postbyte initial :

- 00 pour le registre X
- 20 pour le registre Y
- 40 pour le registre U
- 60 pour le registre S

INTRO5 (partie CH POST) après différentes vérifications ajoute l'offset au postbyte initial et range le résultat à la place de ce postbyte pour constituer le postbyte définitif incluant à la fois la valeur du déplacement et la désignation du registre concerné X, Y, U ou S.

$$\begin{array}{r}
 + \quad 0 \text{ R R } 0 \quad 0 \text{ 0 0 0} \quad (\text{postb. initial}) \\
 \quad 0 \text{ 0 0 } * \quad * * * * \quad (\text{offset}) \\
 \hline
 0 * * * \quad * * * * \quad (\text{postbyte final})
 \end{array}$$

Adress	Opération	Commentaire
E454 50	INTRO5 ASLD	
E455 20	DEC	POSITIF
E457 0C	E7	A LDD
E459 A6	84	A L3A
E45B C1	18	A CMPD
E45D 22	05	E459
E45F 40		ASLA
E460 20	07	E459
E462 20	27	E467
E464 40		ISTPDC ASLA
E465 20	E1	E420
E467 20	1E	E467
E469 40	E3	A POPD5 ASL
E46B 20	1A	E467
E46D 40	04	A COMPST LDA
E46F 34	04	A
E471 40	E9	A
E473 47	04	A
E475 20	25	E47C
E477 0C	E7	A POSITIF
E479 A6	04	A
E47B 40		ASLA
E47D 20	CA	E420
E47F 0B	E3	A
E481 24	E9	E44D
E483 50		HEED
E485 CA	1F	A
E487 24	E6	E44E



RR	offset 5 bits
1 R R 0 0 0 0	0 INC+
1 R R 1 0 0 0	1 INC++
1 R R 0 0 0 1	2 DEC-
1 R R 1 0 0 1	3 DEC--
1 R R 1 0 1 0	4 offset nul
1 R R 1 0 1 0	5 offset AccB
1 R R 1 0 1 1	6 offset AccA
1 R R 1 1 0 0	8 offset 8 bits
1 R R 1 1 0 0	9 offset 16 bits
1 R R 1 1 0 1	8 offset AccD
1 X X 1 1 0 0	C offset PCRL0 bits
1 X X 1 1 0 1	D offset PCRL6 bits
1 X X 1 1 1 1	F Indirect Etendu
BINAIRE	HEXA
POSTBYTE	signification



## Organigramme suite EXPOCT fin AFIVAL Parties INTRO8 et INTRO16

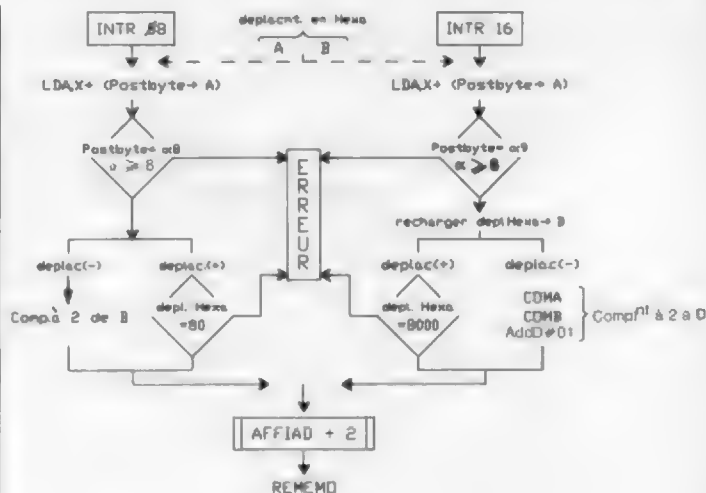
Si le déplacement appartient à l'intervalle  $[-128, +127]$  AFIVAL effectue un branchement à INTR 08 alors que pour un déplacement  $\in [-32768, +32767]$ , c'est à INTR16 que s'effectue le branchement. INTR08 et INTR16 calculent respectivement l'offset sur 8 ou 16 bits à partir du déplacement entrée sur 5 chiffres décimaux à partir du clavier et conversion en Hexa. L'offset est rangé en RAM dans le programme utilisateur immédiatement après le postbyte. Offset et adresse correspondante sont également affichés grâce à AFFIAD + 2.

```

E467 07 E7 A INTR08 LDD :PRELDM
E469 06 06 A LDA .74
E46B 09 09 A DDA .14
E46C 24 00 E426 DEC E426: SI DIFFERENT BITS, CAPTEUR
E46E 06 06 A DDA .14
E46F 04 04 A DDA .04
E471 01 01 A DDA .01 POST OCTET 00115"
E473 26 01 E426 DNE E426:
E475 08 08 A ASL PUSHBMS OUT POSITIF"
E477 24 00 E402 DEC STACKEN 201: STOCKER COMME
E479 06 06 A DDA .14
E47B 07 07 A STACKEN STB .7
E47C 17 07AE E7C0 LDB AF(100+2)
E47F 16 07AE E120 LDBA REVERSED
E482 01 01 A STACKEN CMPD 0001
E484 27 00 E426 DEC E426:
E486 24 02 E41A DDB STACKEN
E488 06 06 A INTR16 LDD .74
E48A 09 09 A DDA .14
E48B 24 00 E426 DEC E426: DIFF 16BITS
E48D 06 06 A DDA .14
E48E 04 04 A DDA .04
E490 01 01 A CMPA 0000 16BITS"
E492 26 02 E426 DNE E426:
E494 0C 07 A LDD :PRELDM
E496 08 07 A ASL :PUSHBMS POSITIF"
E498 24 09 E483 DEC COMP20 OUT
E49A 03 03 A DDB .03
E49B 27 07 CMPD 0000
E49C 03 0001 A ADDD 0001
E49E 0B 04 A CMDB16 STB .7
E4A1 20 07 E47C DDBA STACKEN+2
E4A3 1003 0009 A COMP20 CMPD 000000
E4A7 27 00 E484 DEC AC1
E4A9 28 04 E49F DDBA CMDB16
  
```

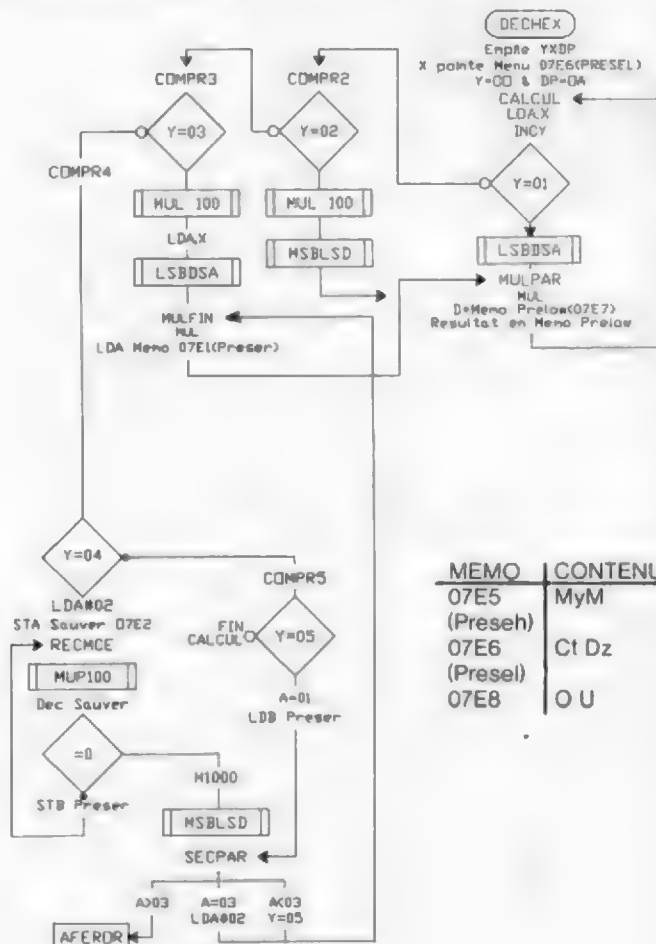
INTRO 8 & 16

POSTBYTE	HEXA	signification
0 R R 0 0 0 0 0	0	offset 5 bits
1 R R 0 0 0 0 1	1	INC+
1 R R 0 0 0 1 0	2	DEC-
1 R R 1 0 0 1 1	3	DEC--
1 R R 1 0 1 0 0	4	offset nul
1 R R 1 0 1 0 1	5	offset AccB
1 R R 1 0 1 1 0	6	offset AccA
1 R R 1 1 0 0 0	8	offset 8 bits
1 R R 1 1 0 0 1	9	offset 16 bits
1 R R 1 1 0 1 1	B	offset AccD
1 X X 1 1 1 0 0	C	offset PCR8 bits
1 X X 1 1 1 0 1	D	offset PCR16 bits
1 X X 1 1 1 1 1	F	Indirect Etendu



## Conversion DEC → HEXA

Ce sous-programme de conversion appelé par AFIVAL convertit le déplacement My MCtDU, en Hexa et place le résultat en Memos 07E7, E8.



MEMO	CONTENU
07E5 (Preseh)	MyM
07E6 (Prese1)	Ct Dz
07E8	O U

E4F4 7A	30	A	DECHET PMS	1,1,DP	
E4F6 8A	0A	A	LDA	000A	
E4F8 1F	80	A	TFB	4,DP	1, 05 34
E4FA DE	07EA	A	LDA	07EAE1	
E4FC 10BE	A660	A	LDA	0050	
E500 A6	0A	A	CALCUL	LDA	1,0
E502 71	21	A	LEA1	1,7	
E504 190C	0001	A	COMP	0001	
E506 26	30	E516	ONE	COMP2	
E508 00	06	E573	USD	4,000A	100 DANS ACEA
E50A 3D		MULPAR	MUL		
E50E F5	07E7	A	ARGO	PRELON	4-PRELON 05 D
E510 FB	07E7	A	STB	PRELON	
E512 20	ED	E501	0A4	CALCUL	
E514 100C	0002	A	COMP2	0002	
E516 26	06	E522	ONE	COMP2	
E518 08	60	E500	0SP	MUL:0C	MUL DES CENTAINES
E51A 00	A2	E50C	0SP	MUL:0B	
E51C 21	ED	E506	0PA	MUL:0A	
E51E 100C	0003	A	COMP2	0003	
E520 26	0C	E534	ONE	COMP2	
E522 00	61	E500	0SP	MUL:0A	MUL DES MILLIERS
E524 A6	0A	A	LDA	1,1	
E526 00	45	E573	0SP	1,000A	
E528 20		MULFIN	MUL		
E52A 06	07E1	A	LDA	PRESEN	
E52C 20	34	E500	0A0	MUL:0A	
E52E 100C	0004	A	COMP2	0004	
E530 26	23	E500	ONE	COMP2	
E532 06	02	A	LDA	0002	
E534 07	07E2	A	STA	SAUVER	
E536 00	50	E501	RECICE	000	MUL DES DIZAINES III MILLIERS
E538 1A	0712	A	DEC	SAUVER	
E53A 27	05	E540	0ED	010000	
E53C F7	07E1	A	STB	PRESEN	
E53E 20	14	E52F	0A4	RECICE	
E540 00	35	E502	010000	000	
E542 00	03	A	SECPAP	0002	
E544 24	0A	E557	0A0	TESTED	
E546 100C	A665	A	LDA	0005	
E548 26	07	E52E	0A0	MULFIN	
E54A 22	11	E560	TESTER	0001	
E54C 06	02	A	LDA	0002	
E54E 20	81	E52E	0A0	MULFIN	
E550 100C	0005	A	COMP2	0005	
E552 26	0E	E571	ONE	FIRCAL	
E554 06	01	A	LDA	0001	
E556 F6	07E1	A	LDA	PRESEN	
E558 21	E2	E540	0A0	SECPAP	
E55A 33	78	A	RECPAP	0A0	0P, 1, Y, U
E55C 1F	12	A	TFB	1, Y	
E55E 16	0260	E701	LDA	CLERO	
E560 29	00	A	FIRCAL	PULS	FE, 1, Y, DP

E560 33	00	3900	MUL100	LEA1	INTER3, PEP
E562 24	00	A	P045	U	
E564 1F	00	A	MUP100	TFB	BP, 0
E566 3F	90	A	TFB	D, A	
E568 20			MUL		
E56A 29			PTS		
E56C 27	07E1	A	INTER3	STB	PRESEN
E56E 29			PTS		

MUL100  
PC pile = E597  
MUP100  
OAXOA  
64 → B  
RTS\*

\*Si entrée en MUL100  
branchement en E597  
lors du désempilage

STB Memo Preser 07E1  
RTS

E572 C6	00	A	LSB00A	1,1	0000
E574 40			DECENC	LSA	
E576 5A			DEC		000A
E578 0A	0A	A	COMP		000A
E57A 26	FA	E575	ONE	DECENC	
E57C 44			DEC715	LSA	
E57E 5A			DEC		000A
E580 26	FE	E579	ONE	DEC715	
E582 1F	0P	A	TFB	DP, 0	
E584 29			PTS		

$A = \alpha\beta \rightarrow O\beta = A$   
 $\downarrow$   
 $DP = OA \rightarrow B$   
 $\downarrow$   
RTS

E582 A6	0A	A	MUL:0A	LDA	1,1
E584 44			LDA		
E586 44			LDA		
E588 44			LDA		
E58A 44			LDA		
E58C 20	1F	A	LEA1	1,1	
E58E 29			RTS		

LDA, X  
 $\downarrow$   
 $A = \alpha\beta \rightarrow O\alpha = A$   
 $\downarrow$   
DEC X  
 $\downarrow$   
RTS

## Interface avec un magnétophone à cassette

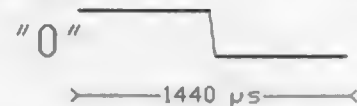
### Généralités

L'interface avec un magnéto à cassette qui se trouve sur la partie clavier/visualisation donne à l'utilisateur la possibilité de sauver et de relire ses programmes sur un magnétophone ordinaire. Le standard d'enregistrement utilise une modulation de durée (P.D.M.). L'enregistrement a les caractéristiques suivantes :

1) Un «1» logique est traduit par une période de signal rectangulaire de # 360  $\mu$ s



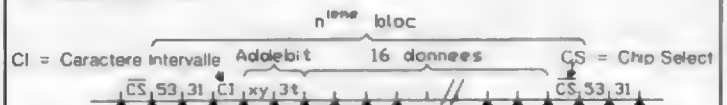
2) Un «0» logique est traduit par une période de signal rectangulaire de # 1440  $\mu$ s. [La durée d'un «0» est quatre fois plus importante que celle d'un «1».]



3) Un octet est enregistré sous la forme d'un «0» suivi des 8 bits significatifs de l'octet proprement dit et d'un «1» d'identification de fin.

4) Les données du programme à sauvegarder sont regroupées par 16 octets dans des blocs consécutifs de 22 caractères. Les 1<sup>er</sup> et dernier blocs peuvent comporter moins de 16 octets de données. En effet, si le programme RAM à transférer commence par exemple à l'adresse 0105, le 1<sup>er</sup> bloc comportera les données d'adresse 0105 à 010F, le 2<sup>e</sup> bloc les données d'adresse 0110 à 011F, etc

5) La structure d'un bloc est la suivante : Caractère 53, caractère 31, caractère intervalle, MSB Adresse début bloc, LSB, 16 datas, caractère de checksum. Pour le dernier bloc, le caractère 31 est remplacé par le caractère 4A.



Il faut ajouter 03 au nombre de données pour composer le caractère intervalle (2 octets d'adresse début bloc et 1 octet de checksum). L'adjonction d'un caractère checksum permet de vérifier la bonne qualité de la transmission lors de la réception.

6) Deux caractères FF précèdent la transmission du 1<sup>er</sup> bloc.

Le programme de transfert [données → magnétophone] est appelé lorsque l'utilisateur appuie sur la touche PUNCH. Le programme de transfert inverse [données enregistrées sur cassette → Mémoire RAM] est appelé à partir de la touche LOAD. Nous analyserons successivement ces deux programmes dans les paragraphes suivants.

## Chargement d'un programme d'une zone mémoire vers un magnétocassette

E641 0E	007D	A	EXPUNC	L01	00007D	
E644 0F	FE	A	STL	<DISBUF+4	AFFICHER 3 POUR START	
E646 17	F04C	E0B3	L0B0	0AB00	FABRIQUER ADRESSE DE DEBUT	
E649 0F	F2	A	STL	<SAB001	SAUVE DEBUT ADRESSE	
E64B 06	09	A	L0A	0009	AFFICHER 1 POUR FIN	
E64D 07	3F	A	STA	<DISBUF+5		
E64F 17	F043	E0B3	L0B0	0AB00	FABRIQUER ADRESSE DE FIN	
E652 0F	F4	A	STL	<SAB002	SAUVE FIN ADRESSE	
E654 0F			CLR0		ACCES A 0000	
E655 07	0007	A	STA	SEMCNT		
E65B 0A			DECA		PR EN SORTIE	
E657 07	0005	A	STA	DISCNT		
E65C 06	04	A	L0A	0004	ACCES A 000 DU PIA	
E65E 07	0007	A	STA	SEMCNT		
E661 0F			CLR0		LIGNE No 0 0 0 ET P06 = 0	
E662 07	0005	A	STA	DISCNT		
E665 06	FF	A	L0A	00FF		
E667 0D	71	E0B6	B0R	01500	FABRIQUATION D'UNE SERIE	
E669 0D	0F	E0D0	B0R	01500	DE 10 PERIODES DE 1400 ET 5000HZ	
E66B 0D	03	E0B9	B0R	00000	CHARGER CARAC DE DEBUT	
E66D 06	FF	A	L0A	00FF		
E66F 06	F3	A	L0B	<SAB001+1	P000 FAIBLE ADRESSE DEBUT	
E671 04	04	A	PS00	0		
E673 06	06	A	S0B0	0	A - B SANS ACCA	
E675 04	0F	A	COBFEN	000A	000F	
E677 08	03	A	AB00	0003	TRANSMISSION D'UN CARACTERE	
E679 0D	3F	E0B6	B0R	01500	INTERVALE	
E67B 07	F1	A	STA	<SAB000		
E67D 0E	07F2	A	L03	<SAB001	DEBUT ADRESSE	
E680 0D	5A	E0B0	B0R	00000	TRANSMISSION MSD ADRESSE DEBUT	
E682 06	F1	A	AB00	<SAB000		
E684 07	F1	A	STA	<SAB000		
E686 20	03	A	LE00	1,1		
E68B 0D	4E	E0B0	B0R	00000	TRANSMISSION LSD ADRESSE DEBUT	
E68D 06	F1	A	AB00	<SAB000		
E68F 07	F1	A	STA	<SAB000		
E690 0E	F2	A	L03	<SAB001	1 POINTE SUR PROGRAMME A ENREGISTRER	
E692 0D	40	E0B0	B0R	00000	TRANSMISSION DES DONNEES	
E694 06	F1	A	AB00	<SAB000		
E696 0C	F4	A	CMF0	<SAB002	FIN D'ENREGISTREMENT ?	
E698 27	0A	E0A4	B0R	00000	OUT, TRANSMETTRE FIN	
E69A 20	03	A	LE00	1,1	NON, DONNEE SUIVANTE A TRANSMETTRE	
E69C 0F	F2	A	ST0	<SAB001	SAUVE POINTEUR	
E69E 06	F3	A	L0A	<SAB001+1		
E6A0 05	0F	A	B01A	000F	16 CARACTERES TRANSMIS ?	
E6A2 26	EC	E0B0	B0R	00000	NON, CONTINUER TRANSMISSION	
E6A4 06	F1	A	FINENR	L0A	<SAB000	OUT, 16 CARACTERES TRANSMIS
E6A6 00			MEM0			
E6A7 0D	31	E0B0	B0R	01500	TRANSMISSION CODE POUR 16 CARAC TRANSMIS	
E6A9 0C	F4	A	CMF0	<SAB002	FIN PROGRAMME ?	
E6AB 27	10	E0C3	B0R	00000	OUT, TRANSMETTRE FIN CHARGEMENT	
E6AD 06	F2	A	L0A	<SAB001	NON, MSD DEBUT ADRESSE	
E6AF 01	F4	A	CMF0	<SAB002	< MSD FIN ADRESSE ?	
E6B1 20	00	E0A0	B0R	00000	OUT, RECOMMENCER CYCLE	
E6B3 06	F3	A	L0B	<SAB001+1	NON, LSD DEBUT ADRESSE	
E6B5 06	F5	A	L0A	<SAB002+1	LSD FIN ADRESSE (?) ?	
E6B7 04	F0	A	AND0	00F0		
E6B9 24	04	A	PS00	0		
E6BB 01	E0	A	CMF0	0	COMPARE B A ACCA	
E6BD 26	AC	E0B0	B0R	00000	OUT, RECOMMENCER UN CYCLE	
E6BF 0D	0F	E0B0	B0R	00000	NON, TRANSMETTRE DEBUT CHARGEMENT	
E6C1 06	F5	A	L0A	<SAB002+1	DU CODE DE FIN	
E6C3 20	00	E075	B0R	00000	CHARGEMENT PUIS FIN	

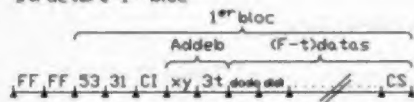
■  $\alpha \cdot \beta \rightarrow T$

Signifie transmission de  $\alpha \cdot \beta$

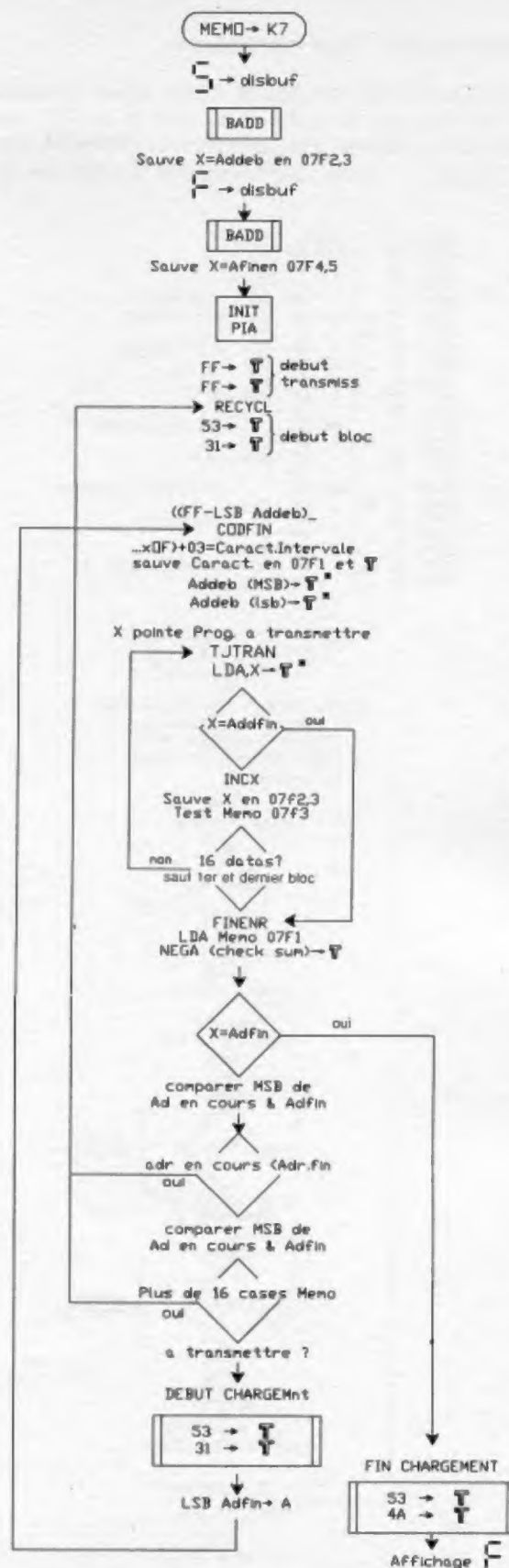
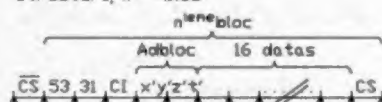
■  $\alpha \beta \rightarrow T$

Signifie transmission de  $\alpha \beta$  et addition de  $\alpha \beta$  a Memo 07F1 pour elaborer checksum

■ structure 1<sup>er</sup> bloc



■ Structure, n<sup>eme</sup> bloc



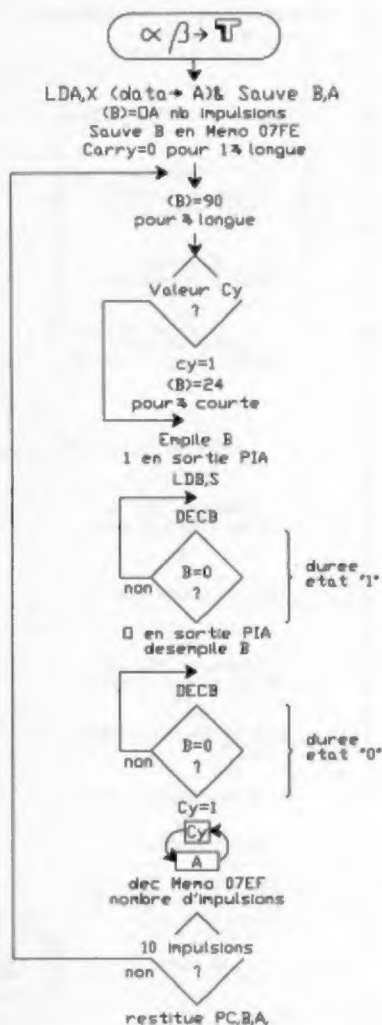
## Transmission d'un caractère

Les différents bits constitutifs d'une valeur hexadécimale sont transmis sur la sortie 6 du port B du PIA selon un codage MIL grâce au sous-programme TRANSM. Ce s.p. a été noté  $\alpha\beta$  dans l'organigramme général EXLOAD.

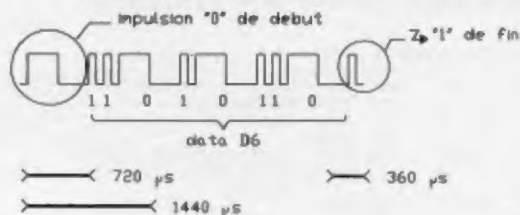
```

E600 A6 04 A TRANSM LDA .F
E601 34 06 A DLYPMS PMS B,A
E602 C6 06 A LDB #006
E603 07 EF A STD /SAVES
E604 1C FE A ANCC #FFE METTRE CARRY A 0
E605 C6 10 A DOUCL3 LDB #990 DELAI = 72MICROSEC
E606 24 02 E60B BCC SAUT
E607 C6 24 A LDB #024 DELAI = 18MICROSEC
E608 34 04 A SAUT PMS B
E609 C6 40 A LDB #000
E60A 07 AD05 A STD DISCNT PMS = 1
E60B E6 04 A LDB #8 NOUVEAU DELAI 100 OU 72MICROSEC
E60C 5A DOUCL3 DECB DELAI 100 OU 72MICROSEC
E60D 26 F0 E6F1 BNE DOUCL1
E60E F7 AD05 A STD DISCNT PMS = 0
E60F 32 04 A PULS B
E610 5A DOUCL3 DECB NOUVEAU DELAI 100 OU 72MICROSEC
E611 26 F0 E6F1 BNE DOUCL2
E612 1A 01 A ANCC #001 METTRE CARRY A 1
E613 46 00A BORA
E614 06 EF A DEC /SAVES
E615 26 BF E6E2 BNE DOUCL3 TERMINER DELAI DE 5MS
E616 35 06 A PULS PC,B,A

```



• Ex : transmission de la valeur hexadécimale D6.



## Chargement d'un programme provenant d'une cassette

```

E609 CC 0000 A EXLOAD LDA #000000
E60A FD AD06 A STD SCMBEG ACCES A 00006
E60B CC #FF A LDB #0FF7F
E60C FD AD04 A STD B1SREG PA ET PB EN SORTIE
E60D CC #004 A LDB #004004 P07 EN ENTREE
E60E FD AD06 A STD SCMBEG ACCES A 00AD
E60F CC #FFA A LDB #0FF9A ETEINDRE LES AFFICHEURS
E610 FD AD0A A STD B1SREG ET SELECTIONNER LE 1ier B1611
E611 00 00 ESC3 DETECS B0R B1TLS DETECTE CARACTERE DEBUT CHARGEMENT
E612 C1 53 A C0P0 #057 CARACTERE 5 TRANSMIS ?
E613 26 FA E6F1 BNE DETECS MEN,CONTINUER A CHERCHER CARACTERE
E614 00 CA ESC3 B0R B1TLS OUI, CARACTERES SUIVANTS
E615 C1 31 A C0P0 #051 CARACTERE 1 TRANSMIS ?
E616 27 0B E605 B0R C0P001 OUI, CARACTERES SUIVANTS
E617 C1 4A A C0P0 #00A NON, CARACTERE FIN = 1 ?
E618 26 F0 E6F1 BNE DETECS NON, DETECTER CARACTERE DE FIN
E619 06 07 A AFICHA LDA #007 OUI, AFFICHER FIN DU CHARGEMENT
E620 32 E637 B0R B1SFIN

```

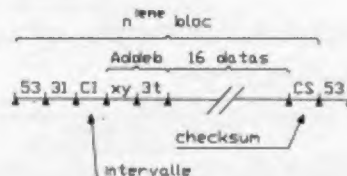
## Mise en mémoire, pointée par X des caractères transmis

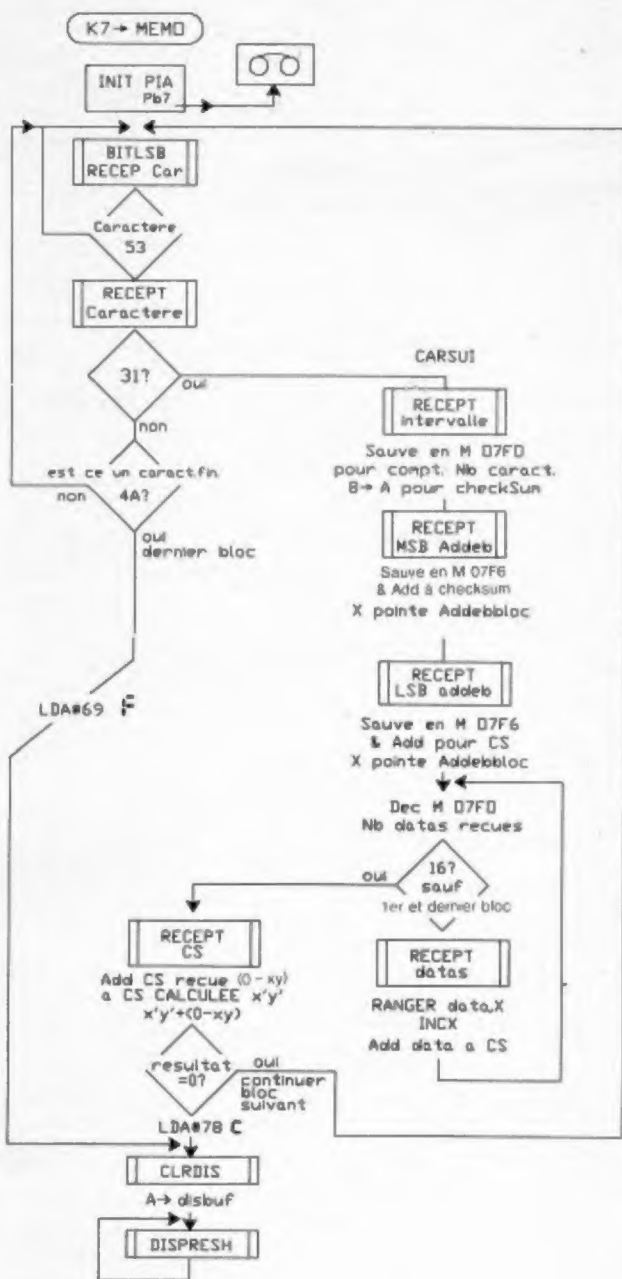
```

E605 00 0C ESC3 C0P001 B0R B1TLS CONVERSION INTERVALE
E606 07 F0 A STD /SAVCM1
E607 06 F0 A LDB /SAVCM1
E608 00 06 ESC3 B0R B1TLS
E609 07 F0 A STD /SAV01
E60A 34 04 A PMS B
E60B AD E0 A ADDA #5+ A + 5 DANS ACCA
E60C 0A F0 A DEC /SAVCM1
E60D 00 AC ESC3 B0R B1TLS
E60E 07 F7 A STD /SAV02
E60F 34 06 A PMS B
E610 AD E0 A ADDA #5+ A + 5 DANS ACCA
E611 06 F0 A LDB /SAV01 X CONTIENT ADRESSE DE CHARGEMENT
E612 0A F0 A SUBCHA DEC /SAVCM1
E613 27 0A E62B B0R C0P000 DERNIERE ADRESSE ?
E614 00 0E ESC3 B0R B1TLS NON, CONTINUER A CHERCHER
E615 E7 00 A STD #1+ ET A METTRE EN MEMOIRE
E616 34 06 A PMS B
E617 AD E0 A ADDA #5+ A + 5 DANS ACCA
E618 20 F2 E61F B0R SUBCHA OUI, DERNIERE ADRESSE
E619 34 04 A PMS B
E620 AD E0 A ADDA #5+ ERREUR DANS LA TRANSMISSION
E621 27 0C ESC3 B0R B1TLS OUI, AFFICHER L'ERREUR
E622 17 FCME E200 DISFIN L0R5 CLAB15
E623 07 FA A STA /DISAUF
E624 17 FAC3 E67B B0PFIN L0R5 DISPRE
E625 20 F0 E63C B0R B0PFIN

```

• Structure nième bloc

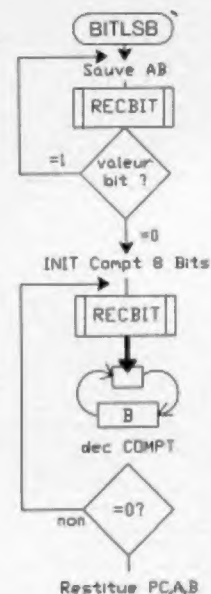
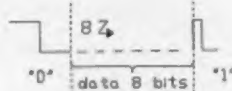




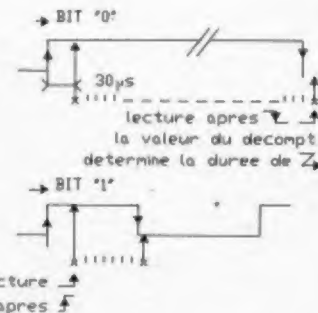
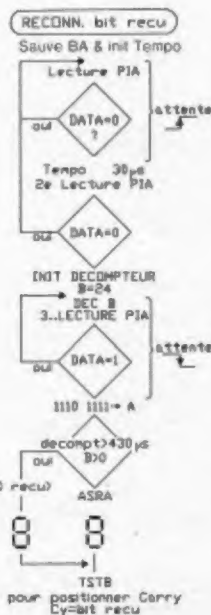
## Réception caractère

ES03 34	02	A	BITLSB PING	A	
ES05 00	04	ES09	BITCAL BSA	RECBIT	RECONNAISSANCE BIT TRANSMIS
ES07 20	FC	ES05	BPL	BITCAL	BIT CARAC. = 1
ES09 06	08	A	LBA	#00	RECONNAISSANCE BITS SUIVANTS
ES0B 00	CE	ES09	COMPT BSA	RECBIT	BIT CARAC=0 RECON BITS SUIVANTS
ES0D 1C	FE	A	ANBCC	#FE	CARRY=0
ES0F 20	02	ES05	#02	BITCAL	BIT CARACTERE = 0 ?
ES01 1A	03	A	ORCC	#001	NON, METTRE CARRY A 1
ES03 56		BITCAL	ANB		OUI, DEPLACER LA CARRY
ES04 44		DECA			PAR ROTATIONS SUCCESSIVES
ES07 26	F4	ES0B	IME	COMPT	CARRY DANS LSB DE ACCB
ES07 75	02	A	PULS	PC, A	

## Structure caractere



ES09 24	06	A	RECBIT PSBS	B, A
ES10 C6	04	A	MOULTY LBO	#004
ES0F 06	AD05	A	SCARRY LBA	DISCMT
ES02 40			LSLA	CHARGER PBT PIA
ES07 24	FA	ES0F	BCC	SCARRY
ES05 56		OLY30U	RECB	PBT=1?
ES06 26	FD	ES05	IME	RECBIT
ES08 06	AD05	A	LBA	DISCMT
ES08 40			LSLA	CHARGER DE NOUVEAU PBT
ES0C 24	EF	ES0D	BCC	MOULTY
ES0E C6	24	A	LBO	#024
ES09 56		CARRY	DECB	NON, NOUVELLE ATTENTE D'UN CARAC.
ES01 06	AD05	A	LBA	DISCMT
ES04 40			LSLA	OUI, DELAI 450MICROSEC MINI.
ES05 25	F8	ES0B	BCC	CARRY
ES07 06	EF	A	LBA	#FEF
ES0F 50			TSTB	TOUJOURS 1?
ES0A 20	01	ES0B	DMI	AFSIGN
ES0C 47		ASPA		OUI, AFFICHER SIGNE POUR 0
ES0D 07	AD04	A	AFSIGN	STA
ES0E 50			TSTB	DISSREG
ES01 33	06	A	PULS	PC, B, A





# Table des matières

<b>Chapitre I :</b>	
<b>De la logique câblée au microprocesseur.....</b>	<b>3</b>
— Introduction au microprocesseur	
• L'être physique : l'homme	
• L'être physique : le robot	
• Constitution d'une «entité» intelligente	
— L'UAL du MOPET	
• Fonctions logiques	
• Fonction mémoire	
• Fonction arithmétique	
• Décodage de l'instruction	
• Le séquenceur	
— Le cheminement des informations dans une unité centrale	
— Sélection mémoire	
— Le 6809	
<b>Chapitre II :</b>	
<b>Un petit système d'initiation :</b>	
<b>le Microkit 09.....</b>	<b>29</b>
— Description de la carte unité centrale et de la carte clavier	
— Montage de la maquette	
— L'architecture interne du 6809	
— Programmation - Utilisation de la maquette	
— Les instructions du 6809	
— Les flags	
— Les modes d'adressage	
<b>Chapitre III :</b>	
<b>Rôle des interruptions matérielles et logicielles.....</b>	<b>61</b>
— Les interruptions	
• Interruptions matérielles	
• Interruptions logicielles	
— Les instructions d'interruptions	
<b>Chapitre IV :</b>	
<b>Aspects du logiciel.....</b>	<b>73</b>
— Présentation générale	
— Mise en route et initialisation	
— Affichage	
— Allumage des afficheurs	
— Clavier	
— Examen et changement du contenu mémoire	
— Examen et changement du contenu des registres	
— Calcul automatique d'offset	
— Interface avec un magnétophone à cassette	